Title

Succinct Non-Interactive Arguments for Arithmetic Circuits

Author

Spooner, Nicholas

Publication Date

2020

Peer reviewed|Thesis/dissertation

Succinct Non-Interactive Arguments for Arithmetic Circuits

by

Nicholas Perry Spooner

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alessandro Chiesa, Chair
Professor Shafi Goldwasser
Professor Kenneth A. Ribet

Fall 2020

Succinct Non-Interactive Arguments for Arithmetic Circuits

Nicholas Perry Spooner

Abstract

Succinct Non-Interactive Arguments for Arithmetic Circuits

by

Nicholas Perry Spooner
Doctor of Philosophy in Computer Science

University of California, Berkeley
Professor Alessandro Chiesa, Chair

This thesis describes a family of new constructions of "succinct" non-interactive arguments (SNARGs) for arithmetic circuit satisfiability. An argument is a protocol by which a prover can convince a verifier of the truth of some statement; in this work, the statement will be of the form "there exists w such $C(x, w) = 1$", where C is an arithmetic circuit. By "succinct", we mean that the communication is polylogarithmic in the size of C.

All of these constructions are unconditionally secure in the random oracle and quantum random oracle models. In particular, they do not require any private setup. This is achieved in each case by designing an *interactive oracle proof* and then applying a transformation of Ben-Sasson, Chiesa and Spooner. We show that our argument systems are both asymptotically efficient and feasible in practice, demonstrating the usefulness of this approach.

More specifically, we obtain the following.

1. A succinct non-interactive argument (AURORA) for general arithmetic circuits, with verification time linear in the size of the circuit.

2. A succinct non-interactive argument for "structured" arithmetic circuits, with verification time *polylogarithmic* in the size of the circuit.

3. A succinct non-interactive argument with preprocessing (FRACTAL) for general arithmetic circuits, where the verification time (after offline preprocessing) is *polylogarithmic* in the size of the circuit.

_____
Professor Alessandro Chiesa
Dissertation Committee Chair

To Mum and Dad. *(Don't worry, you don't have to read it.)*

Contents

List of Figures

Acknowledgments

One man deserves the credit;
one man deserves the blame.
Nikolai Ivanovich Lobachevsky is his name!

Tom Lehrer, *Lobachevsky*.

If one man deserves the credit (or blame) for the existence of this thesis, that is my advisor, Alessandro Chiesa.[1] His enduring support, enthusiasm, and patience have been absolutely invaluable throughout the six years we have worked together. Over this period he has dedicated an enormous amount of time and energy to mentoring me, guiding me both in my research and in my personal development as an academic. I only hope that I will be as good an advisor to my own students as Ale has been to me.

Of course, no researcher exists in a vacuum, and I have been very fortunate to work closely with many other fantastic co-authors: Eli Ben-Sasson, Cecilia Boschini, Benedikt Bünz, Jan Camenisch, Michael Forbes, Lior Goldberg, Ariel Gabizon, Tom Gur, Peter Manohar, Pratyush Mishra, Dev Ojha, Max Ovsiankin, Michael Riabzev, Madars Virza and Nick Ward. I am particularly indebted to Eli for hosting me twice at Technion, and to Jan for supervising my internship at IBM Research Zürich.

One of the greatest things about the Berkeley theory group is the other graduate students, all of whom are (in my experience) both talented researchers and a fun bunch. Special thanks go to my cohort: Arun, Chinmay[2], Elizabeth, Morris, Rachel and Tarun. Sharing an office with friends is a blessing, as are Bobby G's trivia nights (and Arthur ♡). Thanks also to Seri for his sage wisdom, and to Sam for finally getting the group a coffee machine (and introducing me to *bike party*).

My life as a PhD student started at the University of Toronto, and it would be remiss of me not to thank Toni Pitassi; she is an excellent advisor who fosters the best in her students, even if that means they end up transferring to Berkeley. I had a fantastic time as part of the UofT theory group, thanks in no small part to Noah, Robert, Lalla and Akis, who brightened our nearly-windowless office even in the darkest Canadian winter.

One advantage of academic life is that you get to have friends in lots of places. Writing down a list of all the people who have had a positive impact on my life so far would be impossible, but here is an excerpt: thanks to Jason, Mark, Tom, Alex, Ning, Subin, Matt, Julia, Melissa, Heidi, Sunoo, Mate, Helen, Pedro, Anya, Yolanda, Sybil, Ashish, Jesse, Michael, Anna, Joanna, and Jaimie.

A very special thanks to Zoe, who has saved me from working too hard on many occasions, who bore patiently our various ridiculous adventures around the Bay Area, and who is generally a wonderful, kind and caring person. Thanks also to Punchy, who once saved me from working too hard by closing my TeX editor with her nose.

[1] It should be noted that, to the best of my knowledge, neither he, nor I, nor the real Nikolai Lobachevsky, have ever committed plagiarism.

[2] Chinmay suggested an alternative title for my thesis. I will not say what it was.

I would not be where I am without the support of my family. Thanks to Rosie, Chris and Odie, and to my late granddad Ken, who really wanted one of his grandchildren to be a scientist.[3] Finally, there is not space enough in this entire thesis to account for how much I owe to my mum and dad; they have been behind me every step of the way, and I am immensely grateful for that.

[3]He once took me to a lecture about radar engineering, where I understood absolutely nothing.

Chapter 1

Introduction

There is perhaps no concept more central to the study of mathematics than that of *proof*. Since Euclid, proof has been the gold standard of evidence for mathematical facts, and the method by which new mathematics is built and communicated. Traditionally, a proof is a rhetorical device; a sequence of instructions to another mathematician which guides them convincingly from premise to conclusion. More recently, the development of formal logic provided the tools to study proofs as mathematical objects in themselves (which laid the foundations for the study of *computation* as a mathematical object). However, the essential notion of a proof as a series of deductions remained unchanged.

Over the past fifty years, computer science has introduced new ideas about what constitutes a proof. One very fruitful perspective is that a proof is a sort of game between two parties, a *prover* and a *verifier*. The prover's aim is to convince the verifier of the truth of some statement; the verifier's aim is to avoid being duped. Variations on this simple idea lead to a deep and beautiful theory of the relationship between computation and proof which underpins computational complexity theory, including the famous P vs. NP question.

In the simplest variant, the prover writes down a proof and passes it to the verifier, who uses an efficient algorithm to decide whether to accept it or not. Two basic properties are required: *completeness*, which asserts that every true statement has a proof, and *soundness*, which asserts that if the statement is false then the verifier will not accept any proof. One characterisation of the class NP is as the set of statements which can be proved in this manner; this also corresponds roughly to the classical notion of proof.

Of course, one might reasonably ask, why restrict ourselves to this particular type of game? What if the verifier is allowed to flip coins, and accept a proof of a false statement with very low probability? The set of statements that can be proved in this way equals the class MA, which is believed to be strictly larger than NP. If we additionally allow the prover and verifier to engage in a *conversation*, we get much more: this is the class IP, which is known to be equal to the set of statements which can be decided in polynomial *space*, a much larger class than even MA.

Cryptographic proofs. Viewing the prover as an adversary allows us to think about proofs as cryptographic objects, and proofs have become indispensable in modern cryptography. Moreover, cryptography suggests further variations on the notion of proof. One very important variant, which will be the central subject of this thesis, is a (cryptographic) *argument*. An argument is a proof whose soundness relies on an assumption that the prover is computationally bounded.

This is a significant departure from the classical definition of proof. Classical (NP, even IP) proofs have the property that the *provenance* of the proof is irrelevant: a proof etched on a stone tablet by a mysterious deity is as good as a proof written by a mathematician under close surveillance, so long as it can be verified. The mere *existence* of the proof is enough to convince us; for this reason we sometimes refer to such proofs as 'information-theoretic'. An argument is very different: an all-powerful being might be able to fool us into believing things that are not true. On the other hand, if we have reason to believe that the prover is more mundane, an argument might be perfectly adequate for our purposes.

One might ask, of course: why not always use information-theoretic proofs? The answer is that insisting on such strong soundness limits our ability to obtain proofs with useful properties. Most importantly for this thesis, information-theoretic soundness imposes a lower bound on the *length* of a proof of a given statement (under some reasonable complexity assumptions), even when we allow for randomness and interaction. Arguments, on the other hand, can be much shorter than this bound, a property which enables many important cryptographic applications.

1.1 Contributions of this thesis

This thesis describes a family of new constructions of "succinct" non-interactive arguments (SNARGs) for NP. By "succinct", we mean that the size of the argument is polylogarithmic in the size of the nondeterministic computation being proved.

Our constructions follow the methodology introduced in [32], which builds on the classical SNARG construction of [118]. In [32], it is shown how a certain type of information-theoretic proof system called an *interactive oracle proof* (IOP) can be transformed into a SNARG with unconditional security in the random oracle model.

While each is designed for a different use case, all of the constructions share common roots. One important similarity as that all of them produce proofs for (some variant of) the rank-1 constraint satisfiability (R1CS) problem, a useful generalisation of arithmetic circuit satisfiability. We outline each briefly below.

1. A succinct non-interactive argument (AURORA) for general R1CS instances, with verification time linear in the size of the constraint system.

2. A succinct non-interactive argument for 'structured' R1CS instances, with verification time *polylogarithmic* in the size of the constraint system.

3. A succinct non-interactive argument with preprocessing (FRACTAL) for general R1CS instances, where the verification time (after offline preprocessing) is *polylogarithmic* in the size of the constraint system.

These argument systems are all 'nearly optimal', in an asymptotic sense. For a computation of size N (and fixed security parameter), all of the above systems have proof size $\mathrm{polylog}(N)$, and the prover runs in time $O(N \log N)$. What is optimal for verification depends on the setting. For general instances without preprocessing, the verifier must read the input, and so must run in at least linear time; this is matched by AURORA. For 'structured' instances, where we describe a

computation of size N using $\mathrm{polylog}(N)$ bits of information, we can aim for verification time $\mathrm{polylog}(N)$, as achieved by the second construction. Finally, if we are allowed to preprocess our constraint system into a short cryptographic digest, we can similarly aim for online verification time $\mathrm{polylog}(N)$, as achieved by FRACTAL.

In addition to analysing *asymptotic* efficiency, for both AURORA and FRACTAL we demonstrate concrete efficiency by designing and evaluating prototypes.

The remainder of this chapter proceeds as follows. In Section 1.1.1 we describe and motivate the R1CS problem as the "target" problem for our SNARGs. In Section 1.1.2, we discuss the security guarantees achieved by our constructions.

1.1.1 Rank-1 constraint satisfiability

When building an argument system for NP, an important consideration is the choice of NP-complete problem that the system 'natively' supports. Of course, all NP-complete problems are equivalent under polynomial-time reductions. Yet, whether such protocols can be efficiently used in practice actually depends on: (a) the particular NP-complete problem "supported" by the protocol; (b) the concrete efficiency of the protocol relative to this problem. This creates a complex tradeoff.

Simple NP-complete problems, like boolean circuit satisfaction, facilitate simple argument systems; but reducing the statements we wish to prove to boolean circuits is often expensive. On the other hand, one can design argument systems for rich problems (e.g., an abstract computer) for which it is cheap to express the desired statements; but such argument systems may use expensive tools to support these rich problems.

In this thesis we design concretely-efficient argument systems for (variants of) *rank-1 constraint satisfiability* (R1CS), which is the following natural NP-complete problem over a finite field $\mathbb{F}$.

> Given a vector $v \in \mathbb{F}^k$ and three matrices $A, B, C \in \mathbb{F}^{m \times n}$, can one extend v to $z \in \mathbb{F}^n$ ($n \geq k$) such that $Az \circ Bz = Cz$?

Above, and throughout, we use "$\circ$" to denote the entry-wise (Hadamard) product.

We choose R1CS because it strikes an attractive balance: it is expressive, generalising $\mathbb{F}$-arithmetic circuits via a straightforward linear-time reduction and allowing for unbounded fan-in "sum-gates", yet sufficiently structured to simplify protocol design. Moreover, R1CS has demonstrated *strong empirical value*: it underlies real-world systems [80] and there are compilers that reduce to it from program executions (see [149] and references therein). This has led to efforts to standardize R1CS formats across academia and industry [156].

1.1.2 Security properties

All three constructions are obtained by applying the "BCS transformation" [32] to different *interactive oracle proofs* (IOPs). The IOP model is an information-theoretic proof model introduced by [32, 127] as an interactive generalisation of the *probabilistically-checkable proof* (PCP) model [16].

It is shown in [32] that applying the BCS transformation to a sound IOP yields a sound SNARG in the random oracle model. The resulting SNARG is *unconditionally* secure, in the sense that soundness holds against even a computationally unbounded adversary provided the number of queries the adversary can make to the random oracle is (say) polynomially bounded. As with all constructions proven secure in the random oracle model, in order to instantiate the construction in the real world we replace random oracle with a function we believe to be "unstructured enough"; usually this is some keyless hash function like SHA-3.

Zero knowledge. We show that both FRACTAL and AURORA achieve *zero knowledge*. This means that a party receiving a proof of a statement does not learn anything that she could not have computed by herself, except that the statement is true. This is important in many applications, since we often prove statements relating to secret information.

Proof of knowledge. All of our constructions achieve a stronger soundness property known as *knowledge soundness*. At a high level, this means that one can not only prove that a statement is true, but also that one "knows" *why* it is true. This is important for many applications where proofs are used as part of a larger protocol; in particular, it is crucial for the application to incrementally-verifiable computation in Chapter 5. A SNARG with knowledge soundness is known as a SNARG of knowledge (SNARK).

Transparent setup. Since our SNARGs are unconditionally secure in the random oracle model, they automatically achieve a desirable property known as "transparent setup". A setup procedure is *transparent* if it consists only of sampling and publishing randomness; in particular, there is no secret randomness. This is as compared to "trusted setup", where we require a trusted party or protocol to produce and discard secret randomness. Avoiding the need for trusted setup is enormously beneficial in many applications; see Section 1.2.1 for more details.

Post-quantum security. Moreover, it is shown in [67] that security also holds in the *quantum* random oracle model [48]; that is, all of our SNARG constructions achieve post-quantum security. This provides formal evidence to suggest that our constructions remain post-quantum secure after instantiating the random oracle. This is significant, because many of the most efficient constructions of SNARGs rely on pre-quantum assumptions.

1.2 Comparisons with prior work

We present a small survey of related work. Since the landscape of SNARGs is so vast, we are not able to provide a full account here. Instead, we focus on constructions which are closely related to ours; in particular, which admit reasonable quantitative comparisons. First, we discuss one important criterion for categorising SNARGs, and then we compare related work.

1.2.1 Transparent vs. trusted setup

The first succinct argument is due to Kilian [107], who showed how to use collision-resistant hashing to compile any Probabilistically Checkable Proof (PCP) [16, 82, 11, 10] into a corresponding interactive argument. Micali showed how a similar construction, in the random oracle model, yields succinct *non*-interactive arguments (SNARGs) [118]. Subsequent

work showed that Micali's construction preserves a PCP's zero knowledge [104] and proof of knowledge [144] properties. However PCPs remain expensive, and this approach has not led to SNARGs with good concrete efficiency.

In light of this, a different approach was initially used to achieve SNARG implementations with good concrete efficiency [124, 29]. This approach, pioneered in [94, 87, 113, 46], relied on combining certain linearly homomorphic encodings with lightweight information-theoretic tools known as linear PCPs [103, 46, 136]; this approach was refined and optimized in several works [34, 33, 74, 95, 49, 97]. These constructions underlie widely-used open-source libraries [131] and deployed systems [80], and their main feature is that proofs are very short (a few hundred bytes) and very cheap to verify (a few milliseconds).

Unfortunately, the foregoing approach suffers from a severe limitation, namely, the need for a central party to generate system parameters for the argument system. Essentially, this party must run a probabilistic algorithm, publish its output, and "forget" the secret randomness used to generate it. This party must be trustworthy because knowing these secrets allows forging proofs for false assertions. While this may sound like an inconvenience, it is a *colossal* challenge to real-world deployments. When using cryptographic proofs in distributed systems, relying on a central party negates the benefits of distributed trust and, even though it is invoked only once in a system's life, a party trusted by all users typically *does not exist*!

The responsibility for generating parameters can in principle be shared across multiple parties via techniques that leverage secure multi-party computation [31, 52, 53]. This was the approach taken for the launch of Zcash [1], but it also demonstrated how unwieldy such an approach is, involving a costly and logistically difficult real-world multi-party "ceremony". Successfully running such a multi-party protocol was a singular feat, and systems without such expensive setup are decidedly preferable.

Some setup is unavoidable because if SNARGs without *any* setup existed then so would sub-exponential algorithms for SAT [150]. Nevertheless, one may still aim for a "transparent setup", namely one that consists of *public randomness*, because in practice it is cheaper to realize. Recent efforts have thus focused on designing SNARGs with transparent setup, as we discuss in the next section. Note that *all* of our constructions fall into this category.

1.2.2 Implementations of transparent SNARGs

For purposes of comparison with our constructions, we summarize prior work that has both designed *and implemented* transparent SNARGs; see Fig. 1.2 for a table. For a broader discussion of sublinear arguments, we refer the interested reader to the excellent survey of Walfish and Blumberg [149]. [1]

Based on group-theoretic cryptography. *Bulletproofs* [50, 56] proves the satisfaction of an N-gate arithmetic circuit via a recursive use of a low-communication protocol for inner products, achieving a proof with $O(\log N)$ group elements. *Hyrax* [148] proves the satisfaction of a *layered* arithmetic circuit of depth D and width W via proofs of $O(D \log W)$ group elements;

[1] We also note that recent work [19] has used lattice cryptography to achieve sublinear zero knowledge arguments that are plausibly post-quantum secure, which raises the exciting question of whether these recent protocols can lead to efficient implementations.

the construction applies the Cramer–Damgård transformation [75] to doubly-efficient Interactive Proofs [90, 73]. Both approaches use Pedersen commitments, and so are *vulnerable to quantum attacks*. Also, in both approaches the verifier performs *many expensive cryptographic operations*: in the former, the verifier uses $O(N)$ group exponentiations; in the latter, the verifier's group exponentiations are linear in the circuit's witness size. (Hyrax allows fewer group exponentiations but with longer proofs; see [148].)

Based on symmetric cryptography. The "original" SNARG construction of Micali [118, 104] has advantages beyond transparency. First, it is unconditionally secure given a random oracle, which can be instantiated with fast symmetric cryptography.[2] Second, it is known to be post-quantum secure [67]. But the construction relies on PCPs, which remain expensive.

IOPs are "multi-round PCPs" that can also be compiled into non-interactive arguments in the random oracle model [32]. This compilation retains the foregoing advantages (transparency, lightweight cryptography, and plausible post-quantum security) and, *in addition*, facilitates greater efficiency, as IOPs have superior efficiency compared to PCPs [26, 24, 21, 22, 23].

In this thesis we follow the above approach, by constructing zkSNARKs based on three new IOP protocols. Two recent works have also taken the same approach, but with different underlying IOP protocols, which have led to different features. We provide both of these works as part of our library (Section 3.9), and experimentally compare them with our protocol (Section 5.13). The discussion below is a qualitative and asymptotic comparison.

- **Ligero** [9] is a transparent SNARK that proves the satisfiability of an N-gate circuit via proofs of size $O(\sqrt{N})$ that can be verified in $O(N)$ cryptographic operations. The most natural point of comparison is therefore AURORA. As summarized in Fig. 1.1, the IOP underlying Ligero achieves the same oracle proof length, prover time, and verifier time as the AURORA IOP. However, the query complexity (strictly, symbol size) of AURORA is exponentially smaller, at the expense of increasing round complexity from 2 to $O(\log N)$. The arguments that we obtain are still non-interactive; our smaller query complexity translates into shorter proofs (see Fig. 1.2).

- **Stark** [23] is a transparent SNARK for bounded halting problems on a random access machine. Given a program P and a time bound T, it proves that P accepts within T steps on a certain abstract computer (when given suitable nondeterministic advice) via succinct proofs of size $\mathrm{polylog}(T)$. Moreover, verification is *also* succinct: checking a proof takes time only $|P| + \mathrm{polylog}(T)$, which is polynomial in the size of the statement and much better than "naive verification" which takes time $\Omega(|P| + T)$.

  The natural comparison here is with our SNARG for structured R1CS instances, since this also targets succinctly-represented computation. In this context, both our SNARG and the Stark system achieve polylogarithmic verification. However, while Stark achieves $O(T \log T)$ proof length, $\tilde{O}(T \log^2 T)$ prover time, and logarithmic query and round complexity, our SNARG improves on all of these by a factor of $\log T$, achieving linear proof length, $\tilde{O}(T \log T)$ prover time, and *constant* query and round complexity.

[2] Some cryptographic hash functions, such as BLAKE2, can process almost 1 gibibyte per second [14].

- **Supersonic** [58] is a transparent preprocessing SNARK for arithmetic circuits. It differs from the two constructions above, and from ours, in that the transformation from (polynomial) IOP to SNARK uses a polynomial commitment based on the adaptive root assumption in class groups, rather than a Merkle tree commitment as in [32]. This leads to an order of magnitude improvement in proof size versus Fractal, at the cost of a significantly slower prover and loss of post-quantum security.

	protocol type	**round complexity**	**proof length** (field elts)	**query complexity**	**prover time** (field ops)	**verifier time** (field ops)
Ligero	IPCP †	2	$O(N)$	$O(\sqrt{N})$	$O(N \log N)$	$O(N)$
Stark	IOP	$O(\log N)$	$O(N \log N)$	$O(\log N)$	$O(N \log^2 N)$	$O(N)$
Aurora	IOP	$O(\log N)$	$O(N)$	$O(\log N)$	$O(N \log N)$	$O(N)$

Figure 1.1: Asymptotic comparison of the information-theoretic proof systems underlying Ligero, Stark, and Aurora, when applied to an N-gate arithmetic circuit.
† An IPCP [106] is a PCP oracle that is checked via an Interactive Proof; it is a special case of an IOP.

	name	**setup**	**post quantum?**	**argument size** asymptotic	$N = 10^6$	**verifier time**
[94][87] [113][46]...	various	private	no	$O_\lambda(1)$	128 B	$O_\lambda(k)$ †
[154]	ZK-vSQL	private	no	$O_\lambda(d \log N)$	N/A	$O_\lambda(N)$
[148]	Hyrax	public	no	$O_\lambda(d \log N)$ ‡	50 kB	$O_\lambda(N)$
[50] [56]	Bulletproofs	public	no	$O_\lambda(\log N)$	1.5 kB	$O_\lambda(N)$
[58]	Supersonic	public	no	$O_\lambda(\log N)$	10 kB	$O_\lambda(\log N)$ †
[9]	Ligero	public	yes	$O_\lambda(\sqrt{N})$	4.0 MB	$O_\lambda(N)$
[23]	Stark	public	yes	$O_\lambda(\log^2 N)$	3.2 MB	$O_\lambda(N)$
this work	Aurora	public	yes	$O_\lambda(\log^2 N)$	110 kB	$O_\lambda(N)$
this work	Fractal	public	yes	$O_\lambda(\log^2 N)$	160 kB	$O_\lambda(\log^2 N)$ †

Figure 1.2: Comparison of some non-interactive zero knowledge arguments for proving statements of the form "there exists a secret w such that $\mathcal{C}(x, w) = 1$" for a given *explicit* arithmetic circuit $\mathcal{C}$ of N gates (and depth d) and public input x of size k. The table is grouped by "technology", and for simplicity assumes that the circuit's underlying field has size $2^{O(\lambda)}$ where λ is the security parameter. Approximate argument sizes are given for $N = 10^6$ gates over a cryptographically-large field, and a security level of 128 bits; some argument sizes may differ from those reported in the cited works because size had to be re-computed for the security level and N used here; also, [154] reports no implementation.
† Given a per-circuit preprocessing step.
‡ A tradeoff between argument size and verifier time is possible; see [148].

Chapter 2

Technical preliminaries

2.1 Notation

Given a relation $\mathcal{R} \subseteq S \times T$, we denote by $\mathcal{L}(\mathcal{R}) \subseteq S$ the set of $s \in S$ such that there exists $t \in T$ with $(s,t) \in \mathcal{R}$; for $s \in S$, we denote by $\mathcal{R}|_s \subseteq T$ the set $\{t \in T : (s,t) \in \mathcal{R}\}$. Given a set S and strings $v, w \in S^n$ for some $n \in \mathbb{N}$, the *fractional Hamming distance* $\Delta(v,w) \in [0,1]$ is $\Delta(v,w) := \frac{1}{n}|\{i : v_i \neq w_i\}|$.

We denote the concatenation of two vectors u_1, u_2 by $u_1 \| u_2$, and the concatenation of two matrices A, B by $[A|B]$.

All fields $\mathbb{F}$ in this thesis are finite, and we denote the finite field of size q by $\mathbb{F}_q$. We say that H is a *subgroup* in $\mathbb{F}$ if it is either a subgroup of $(\mathbb{F}, +)$ (an additive subgroup) or of $(\mathbb{F} \setminus \{0\}, \times)$ (a multiplicative subgroup); we say that H is a *coset* in $\mathbb{F}$ if it is a coset of a subgroup in $\mathbb{F}$ (possibly the subgroup itself).

2.1.1 Codes

Interleaved codes. Given linear codes $C_1, \ldots, C_m \subseteq \mathbb{F}^n$ with alphabet $\mathbb{F}$, we denote by $\prod_{i=1}^m C_i \subseteq (\mathbb{F}^m)^n \equiv \mathbb{F}^{m \times n}$ the linear "interleaved" code with alphabet $\mathbb{F}^m$ that equals the set of all $m \times n$ matrices whose i-th row is in C_i. If $C_1 = \cdots = C_m$, we write C^m for $\prod_{i=1}^m C_i$. Since the alphabet is $\mathbb{F}^m$, the Hamming distance is taken *column-wise*: for $A, A' \in \mathbb{F}^{m \times n}$, $\Delta(A, A') := \frac{1}{n}|\{j \in [n] : \exists\, i \in [m] \text{ s.t. } A_{i,j} \neq A'_{i,j}\}|$.

The Reed–Solomon code. Given a subset L of a field $\mathbb{F}$ and $\rho \in (0,1]$, we denote by $\mathrm{RS}[L, \rho] \subseteq \mathbb{F}^L$ all evaluations over L of univariate polynomials of degree less than $\rho|L|$. That is, a word $c \in \mathbb{F}^L$ is in $\mathrm{RS}[L, \rho]$ if there exists a polynomial p of degree less than $\rho|L|$ such that $c_a = p(a)$ for every $a \in L$. We denote by $\mathrm{RS}[L, (\rho_1, \ldots, \rho_n)] := \prod_{i=1}^n \mathrm{RS}[L, \rho_i]$ the interleaving of Reed–Solomon codes with rates $\rho_1, \ldots, \rho_n$.

2.2 Polynomials

2.2.1 Representations of polynomials

We frequently move from univariate polynomials over $\mathbb{F}$ to their evaluations on chosen subsets of $\mathbb{F}$, and back. We use plain letters like f, g, h, π to denote *evaluations* of polynomials, and "hatted letters" $\hat{f}, \hat{g}, \hat{h}, \hat{\pi}$ to denote corresponding polynomials. This bijection is well-defined only if the size of the evaluation domain is larger than the degree. Formally, if $f \in \mathrm{RS}[L, \rho]$ for $L \subseteq \mathbb{F}$, $\rho \in (0, 1]$, then $\hat{f}$ is the unique polynomial of degree less than $\rho|L|$ whose evaluation on L equals f. Likewise, if $\hat{f} \in \mathbb{F}[X]$ with $\deg(f) < \rho|L|$, then $f_L := \hat{f}|_L \in \mathrm{RS}[L, \rho]$ (but we will drop the subscript when the choice of subset is clear from context).

2.2.2 The fast Fourier transform

We often rely on polynomial arithmetic, which can be efficiently performed via fast Fourier transforms and their inverses. In particular, polynomial evaluation and interpolation over an (affine) subspace of size n of a finite field can be performed in $O(n \log n)$ field operations via an additive FFT [110]. Because in practice the number of FFTs we perform is important, when discussing complexities we use the notation $\mathrm{FFT}(\mathbb{F}, m)$ for the cost of a single additive FFT (or IFFT) on a subspace of $\mathbb{F}$ of size m.

Remark 2.2.1. Strictly, an additive FFT evaluates a polynomial of degree d on a subspace of size $d + 1$. To evaluate on a larger subspace (of size n), one can run an FFT over each coset of the smaller space inside the larger one at a cost of $\frac{n}{d} \cdot O(d \log d) = O(n \log d)$. We will suppress this technicality when it appears, and upper bound the cost of such an evaluation by an FFT on a subspace of size n.

2.2.3 Special polynomials

Vanishing polynomials. Let $\mathbb{F}$ be a finite field, and $S \subseteq \mathbb{F}$. We denote by $\mathbb{Z}_S$ the unique non-zero monic polynomial of degree at most $|S|$ that is zero everywhere on S; $\mathbb{Z}_S$ is called the *vanishing polynomial* of S. In this work we use efficiency properties of vanishing polynomials for sets S that have group structure.

If S is a multiplicative subgroup of $\mathbb{F}$, then $\mathbb{Z}_S(X) = X^{|S|} - 1$, and so $\mathbb{Z}_S(X)$ can be evaluated at any $\alpha \in \mathbb{F}$ in $O(\log |S|)$ field operations. More generally, if S is a γ-coset of a multiplicative subgroup S_0 (namely, $S = \gamma S_0$) then $\mathbb{Z}_S(X) = \gamma^{|S|}\mathbb{Z}_{S_0}(X/\gamma) = X^{|S|} - \gamma^{|S|}$.

If S is an (affine) subspace of $\mathbb{F}$, then $\mathbb{Z}_S$ is called an *(affine) subspace polynomial*. In this case, there exist coefficients $c_0, \ldots, c_k \in \mathbb{F}$, where $k := \dim(S)$, such that $\mathbb{Z}_S(X) = X^{p^k} + \sum_{i=1}^{k} c_i X^{p^{i-1}} + c_0$ (if S is linear then $c_0 = 0$). Hence, $\mathbb{Z}_S(X)$ can be evaluated at any $\alpha \in \mathbb{F}$ in $O(k \log p) = O(\log |S|)$ operations. Such polynomials are called *linearized* because they are $\mathbb{F}_p$-affine maps: if $S = S_0 + \gamma$ for a subspace $S_0 \subseteq \mathbb{F}$ and shift $\gamma \in \mathbb{F}$, then $\mathbb{Z}_S(X) = \mathbb{Z}_{S_0}(X - \gamma) = \mathbb{Z}_{S_0}(X) - \mathbb{Z}_{S_0}(\gamma)$, and $\mathbb{Z}_{S_0}$ is an $\mathbb{F}_p$-linear map. The coefficients $c_0, \ldots, c_k$ can be derived from a description of S (any basis of S_0 and the shift γ) in $O(k^2 \log p)$ field operations (see [108, Chapter 3.4] and [27, Remark C.8]).

Lagrange polynomials. For $\mathbb{F}$ a finite field, $S \subseteq \mathbb{F}$, $a \in S$, we denote by $L_{S,a}$ the unique polynomial of degree less than $|S|$ such that $L_{S,a}(a) = 1$ and $L_{S,a}(b) = 0$ for all $b \in S \setminus \{a\}$. Note that

$$L_{S,a}(X) = \frac{\prod_{b \in S \setminus \{a\}}(X - b)}{\prod_{b \in S \setminus \{a\}}(a - b)} = \frac{L'_S(X)}{L'_S(a)} \ ,$$

where $L'_S(X)$ is the polynomial $\mathbb{Z}_S(X)/(X - a)$. For additive and multiplicative subgroups S and $a \in S$, we can evaluate $L_{S,a}(X)$ at any $\alpha \in \mathbb{F}$ in $\mathrm{polylog}(|S|)$ field operations. This is because an arithmetic circuit for L'_S can be efficiently derived from an arithmetic circuit for $\mathbb{Z}_S$ [138].

2.3 Proof systems

Much of this work is about *interactive oracle proofs* (IOPs), and variants thereof. In this section we formally define IOPs and describe relevant properties and complexity measures.

2.3.1 Interactive oracle proofs

Interactive Oracle Proofs (IOPs) [32, 127] combine aspects of Interactive Proofs [15, 91] and Probabilistically Checkable Proofs [16, 11, 10], and also generalize the notion of Interactive PCPs [106].

A k-round public-coin IOP has k rounds of interaction. In the i-th round of interaction, the verifier sends a uniformly random message m_i to the prover; then the prover replies with a message π_i to the verifier. After k rounds of interaction, the verifier makes some queries to the oracles it received and either accepts or rejects.

An *IOP system* for a relation $\mathcal{R}$ with round complexity k and soundness error ε is a pair $(\mathbf{P}, \mathbf{V})$, where $\mathbf{P}, \mathbf{V}$ are probabilistic algorithms, that satisfies the following properties. (See [32, 127] for details.)

Completeness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ in the relation $\mathcal{R}$, $(\mathbf{P}(\mathtt{x}, \mathtt{w}), \mathbf{V}(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol with accepting probability 1.

Soundness: For every instance $\mathtt{x} \notin \mathcal{L}(\mathcal{R})$ and unbounded malicious prover $\tilde{\mathbf{P}}$, $(\tilde{\mathbf{P}}, \mathbf{V}(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol with accepting probability at most $\varepsilon(n)$.

Like the IP model, a fundamental measure of efficiency is the round complexity k. Like the PCP model, two additional fundamental measures of efficiency are the *proof length* p, which is the total number of alphabet symbols in all of the prover's messages, and the *query complexity* q, which is the total number of locations queried by the verifier across all of the prover's messages.

We say that an IOP system is *non-adaptive* if the verifier queries are non-adaptive, namely, the queried locations depend only on the verifier's inputs and its randomness. All of our IOP systems will be non-adaptive.

Since the verifier is public coin, its behavior in the interactive part of the protocol is easy to describe. We can therefore think of $\mathbf{V}$ as a randomized algorithm which, given its prior random

messages and oracle access to the prover's messages, makes queries to the prover's messages and either accepts or rejects.

The foregoing division allows us to separately consider the randomness and soundness error for these two phases, which is useful for a more fine-grained soundness-error reduction. Letting $\mathsf{r_i}$ and $\mathsf{r_q}$ be the randomness complexities of interaction and query phases respectively, the quantities ε_i and ε_q satisfy the following relation (for all instances $\mathtt{x} \notin \mathcal{L}(\mathcal{R})$ and malicious provers $\tilde{P}$):

$$\Pr\left[\Pr_{r\leftarrow\{0,1\}^{\mathsf{r_q}}}[\mathbf{V}^{\vec{\pi}}(\mathtt{x}, m_1, \ldots, m_\mathsf{k}; r) = 1] \geq \varepsilon_\mathsf{q} \;\middle|\; \begin{array}{c}(m_1, \ldots, m_\mathsf{k}) \leftarrow \{0,1\}^{\mathsf{r_i}} \\ \pi_1, \ldots, \pi_\mathsf{k} \leftarrow (\tilde{P}, (m_1, \ldots, m_\mathsf{k}))\end{array}\right] \leq \varepsilon_\mathsf{i} \ .$$

That is, the probability that random messages make $\mathbf{V}$ accept with probability at least ε_q (over internal randomness) is at most ε_i. In particular, the overall soundness error is at most $\varepsilon_\mathsf{i} + \varepsilon_\mathsf{q}$. Note that an IOP with $\varepsilon_\mathsf{i} = 0$ is a PCP, an IOP with $\varepsilon_\mathsf{q} = 0$ is an IP, and an IOP with both $\varepsilon_\mathsf{i} = \varepsilon_\mathsf{q} = 0$ is a deterministic (NP) proof.

Given the above, consider a "semi-black-box" example of soundness-error reduction: the interactive phase is run once, and then we repeat the query phase ℓ times with fresh randomness. This yields an IOP with query complexity $\ell \cdot \mathsf{q}$, randomness complexity $\mathsf{r_i} + \ell \cdot \mathsf{r_q}$, and soundness error $\varepsilon_\mathsf{i} + \varepsilon_\mathsf{q}^\ell$, but with the *same* proof length and number of rounds. The running time of the prover is unchanged, and the verifier runs in time $O(\ell \cdot \mathsf{t}_V)$. By comparison, repetition of the entire protocol yields proof length $\ell \cdot \mathsf{p}$ and $\ell \cdot \mathsf{k}$ rounds, for soundness error $(\varepsilon_\mathsf{i} + \varepsilon_\mathsf{q})^\ell$; the prover runs in time $O(\ell \cdot \mathsf{t}_P)$ and the verifier in time $O(\ell \cdot \mathsf{t}_V)$.

Proof of knowledge. The IOP protocols presented in this paper satisfy a stronger notion of soundness called *proof of knowledge*: if a prover algorithm $\tilde{\mathbf{P}}$ convinces the verifier with sufficiently high probability, it is possible to efficiently extract a witness from $\tilde{P}$. In order to give a formal definition, we define the quantity $w_\mathcal{R}(n) := \max\{|\mathtt{w}| \colon (\mathtt{x}, \mathtt{w}) \in \mathcal{R}, |\mathtt{x}| = n\}$, the maximum witness length for an instance of length n.

Proof of knowledge: There exists a probabilistic polynomial-time algorithm E such that for every instance $\mathtt{x}$ and unbounded malicious prover $\tilde{P}$ that makes $\mathbf{V}$ accept with probability μ, $E^{\tilde{P}}(\mathtt{x}, 1^{w_\mathcal{R}(n)})$ outputs $\mathtt{w}$ such that $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ with probability at least $\mu - \varepsilon(n)$.

2.3.1.1 IOPs of proximity

An IOP *of Proximity* extends an IOP the same way that PCPs of Proximity extend PCPs. An *IOPP system* for a relation $\mathcal{R}$ with round complexity k, soundness error ε, and proximity parameter δ is a pair (P, V) that satisfies the following properties.

Completeness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ in the relation $\mathcal{R}$, $(P(\mathtt{x}, \mathtt{w}), V^{\mathtt{w}}(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol with accepting probability 1.

Soundness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ with $\Delta(\mathtt{w}, \mathcal{R}|_\mathtt{x}) \geq \delta(n)$ and unbounded malicious prover $\tilde{P}$, $(\tilde{P}, V^{\mathtt{w}}(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol with accepting probability at most $\varepsilon(n)$.

Efficiency measures for IOPPs are as for IOPs, except that we also count queries to the witness. Namely, if V makes at most $\mathsf{q}_{\mathtt{w}}$ queries to $\mathtt{w}$ and at most q_{π} queries across all prover messages, the query complexity is $\mathsf{q} := \mathsf{q}_{\mathtt{w}} + \mathsf{q}_{\pi}$. Like with IOPs, we divide public-coin IOPPs into an interaction phase and a query phase.

Low-degree testing. For the purposes of this paper, a low-degree test is an IOPP for the Reed–Solomon relation $\mathcal{R}_{\mathrm{RS}} := \{((L, \rho), p) : L \subseteq \mathbb{F}, \rho \in (0, 1], p \in \mathrm{RS}\,[L, \rho]\}$. In this case ε and δ are functions of ρ.

2.3.2 Zero knowledge

The definitions of unconditional (perfect) zero knowledge that we use for IOPs and for IOPPs follow those in [92, 105, 24]. We first define the notion of a view and of straightline access; after that we define zero knowledge for IOPs and for IOPPs in a way that suffices for our purposes.

Definition 2.3.1. *Let A, B be algorithms and x, y strings. We denote by* $\mathrm{View}\,(B(y), A(x))$ *the* **view** *of $A(x)$ in an interactive oracle protocol with $B(y)$, i.e., the random variable $(x, r, a_1, \ldots, a_n)$ where x is A's input, r is A's randomness, and $a_1, \ldots, a_n$ are the answers to A's queries into B's messages.*

Definition 2.3.2. *An algorithm B has* **straightline access** *to an algorithm A if B interacts with A, without rewinding, by exchanging messages with A and answering any oracle queries along the way.*

We denote by B^A the concatenation of A's random tape and B's output when it has straightline access to A. (Since A's random tape could be super-polynomially large, B cannot sample it for A and then output it; instead, we restrict B to not see it, and we prepend it to B's output.)

For IOPs, we consider unconditional (perfect) zero knowledge against bounded-query verifiers.

Definition 2.3.3. *An IOP system $(\mathbf{P}, \mathbf{V})$ for a relation $\mathcal{R}$ is* **(perfect) zero knowledge against query bound** b *if there exists a simulator algorithm S such that for every b-query algorithm $\tilde{\mathbf{V}}$ and instance-witness pair $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$, $S^{\tilde{\mathbf{V}}}(\mathtt{x})$ and $\mathrm{View}\,(\mathbf{P}(\mathtt{x}, \mathtt{w}), \tilde{\mathbf{V}}(\mathtt{x}))$ are identically distributed. (An algorithm is b-query if, on input $\mathtt{x}$, it makes at most $\mathsf{b}(|\mathtt{x}|)$ queries to any oracles it has access to.) Moreover, S must run in time $\mathrm{poly}(|\mathtt{x}| + \mathsf{q}_{\tilde{\mathbf{V}}}(|\mathtt{x}|))$, where $\mathsf{q}_{\tilde{\mathbf{V}}}(\cdot)$ is $\tilde{\mathbf{V}}$'s query complexity.*

For zero knowledge against arbitrary polynomial-time adversaries, it suffices for b to be superpolynomial. Note that S's running time is required to be polynomial in the input size $|\mathtt{x}|$ and the *actual* number of queries $\tilde{V}$ makes (as a random variable) and, in particular, may be polynomial even if b is not. We do not restrict $\tilde{V}$ to make queries only at the end of the interaction; all of our protocols will be zero knowledge against the more general class of verifier that can, at any time, make queries to any oracle it has already received.

For IOPPs, we consider unconditional (perfect) zero knowledge against unbounded-query verifiers.

Definition 2.3.4. *An IOPP system (P, V) for a relation $\mathcal{R}$ is* **(perfect) zero knowledge against unbounded queries** *if there exists a simulator algorithm S such that for every algorithm $\tilde{\mathbf{V}}$ and instance-witness pair $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$, the following two random variables are identically distributed:*

$$\left(S^{\tilde{V},\mathtt{w}}(\mathtt{x})\ ,\ q_S\right) \quad \textit{and} \quad \left(\text{View}\ (P(\mathtt{x},\mathtt{w}), \tilde{V}^{\mathtt{w}}(\mathtt{x}))\ ,\ q_{\tilde{V}}\right)\ ,$$

where q_S is the number of queries to $\mathtt{w}$ made by S, and $q_{\tilde{V}}$ is the number of queries to $\mathtt{w}$ or to prover messages made by $\tilde{V}$. Moreover, S must run in time $\text{poly}(|\mathtt{x}| + \mathsf{q}_{\tilde{\mathbf{V}}}(|\mathtt{x}|))$, where $\mathsf{q}_{\tilde{\mathbf{V}}}(\cdot)$ is $\tilde{\mathbf{V}}$'s query complexity.

2.3.3 Reed–Solomon encoded IOP

We typically first describe IOPs for which soundness only holds against provers whose messages are Reed–Solomon codewords of specified rates and on which certain *rational constraints* hold, and later "compile" them into standard IOPs.[1] This facilitates focusing on a protocol's key ideas, and leaves handling provers that do not respect this restriction to generic tools.

Later in this thesis we consider two incomparable generalizations of RS-encoded IOPs. In Section 4.4, we define *oracle reductions*, which aim to capture a wide class of reductions between protocols in the IOP setting. In the language of oracle reductions, an RS-IOP is a reduction to univariate low-degree testing. In Section 5.4, we define *RS-encoded holographic IOPs* (RS-hIOPs), which generalize RS-IOPs to a preprocessing setting. Nonetheless, we believe the basic definition is informative, since it captures the core idea with minimal additional complications.

Returning to the matter at hand, we first define what we mean by a rational constraint.

Definition 2.3.5. *A* **rational constraint** *is a pair $(\mathcal{C}, \sigma)$ where $\mathcal{C} = (N, D)$, $N \colon \mathbb{F}^{1+\ell} \to \mathbb{F}$, $D \colon \mathbb{F} \to \mathbb{F}$ are arithmetic circuits and $\sigma \in (0, 1]$ is a rate parameter. A rational constraint $(\mathcal{C}, \sigma)$ and an interleaved word $f \in (L \to \mathbb{F})^{\ell}$ jointly define a codeword $\mathcal{C}[f] \colon L \to \mathbb{F}$, given by $\mathcal{C}[f](\alpha) := \frac{N(\alpha, f_1(\alpha), \ldots, f_\ell(\alpha))}{D(\alpha)}$ for all $\alpha \in L$. A rational constraint $(\mathcal{C}, \sigma)$ is* **satisfied by** *f if $\mathcal{C}[f] \in \text{RS}\,[L, \sigma]$.*[2]

A *Reed–Solomon encoded IOP* (RS-encoded IOP) for a relation $\mathcal{R}$ is a tuple $(P, V, (\vec{\rho}_i)_{i=1}^{\mathsf{k}})$, where P and V are probabilistic algorithms and $\vec{\rho}_1 \in (0, 1]^{\ell_1}, \ldots, \vec{\rho}_{\mathsf{k}} \in (0, 1]^{\ell_{\mathsf{k}}}$, that satisfies the following properties.

Completeness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ in the relation $\mathcal{R}$, $(P(\mathtt{x}, \mathtt{w}), V(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol, where the i-th message of P is a codeword of $\text{RS}\,[L, \vec{\rho}_i]$, and V outputs a set of rational constraints that are satisfied with respect to the prover's messages with probability 1.

[1] Rational constraints enable us to capture useful optimizations that involve testing "virtual oracles" implicitly derived from oracles sent by the prover. Such optimizations reduce argument size in the resulting zkSNARKs as discussed, e.g., in [23].

[2] For $\alpha \in L$, if $D(\alpha) = 0$ then we define $\mathcal{C}[f](\alpha) := \perp$. Note that if this holds for some $\alpha \in L$ then, for *any* word f and rate parameter σ, the rational constraint $(\mathcal{C}, \sigma)$ is not satisfied by f; in particular, the completeness condition does not hold.

Soundness: For every instance $\mathtt{x} \notin \mathcal{L}(\mathcal{R})$ and unbounded malicious prover $\tilde{P}$ whose i-th message is a codeword of $\mathrm{RS}[L, \vec{\rho}_i]$, $(\tilde{P}, V(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol wherein the set of rational constraints output by V are satisfied with respect to the prover's messages with probability at most $\varepsilon(n)$.

All RS-encoded IOPs that we consider also satisfy a proof of knowledge property.

Proof of knowledge: There exists a probabilistic polynomial-time algorithm E such that for every instance $\mathtt{x}$ and unbounded malicious prover $\tilde{P}$, whose i-th message is a codeword of $\mathrm{RS}[L, \vec{\rho}_i]$, that makes V output satisfiable rational constraints with probability μ, $E^{\tilde{P}}(\mathtt{x}, 1^{w_{\mathcal{R}}(n)})$ outputs $\mathtt{w}$ such that $(\mathtt{x}, \mathtt{w}) \in \mathcal{R}$ with probability at least $\mu - \varepsilon(n)$.

A useful complexity measure of a Reed–Solomon encoded IOP is its *maximum rate*, which informally is the maximum over the (prescribed) rates of codewords sent by the prover and those induced by the verifier's rational constraints. To formally define it, we need to first introduce the *degree* and *rate* of a circuit.

Definition 2.3.6. *The* **degree** *of an arithmetic circuit $C \colon \mathbb{F}^{1+\ell} \to \mathbb{F}$ on input degrees $d_1, \ldots, d_\ell \in \mathbb{N}$, denoted $\deg(C; d_1, \ldots, d_\ell)$, is the smallest integer e such that for all $p_i \in \mathbb{F}^{\leq d_i}[X]$ there exists a polynomial $q \in \mathbb{F}^{\leq e}[X]$ such that $C(X, p_1(X), \ldots, p_\ell(X)) \equiv q(X)$. Given domain $L \subseteq \mathbb{F}$ and rates $\vec{\rho} \in (0, 1]^\ell$, the* **rate** *of C is $\mathrm{rate}(C; \vec{\rho}) := \deg(C; \rho_1|L|, \ldots, \rho_\ell|L|)/|L|$. (The domain L will be clear from context.) Note that if $\ell = 0$ then the foregoing notion of degree coincides with the usual one (namely, $\deg(C)$ is the degree of the polynomial described by C), and the foregoing notion of rate is simply $\mathrm{rate}(C) := \deg(C)/|L|$.*

A Reed–Solomon encoded IOP $(P, V, (\vec{\rho}_i)_{i=1}^{\mathsf{k}})$ has *maximum rate* $(\rho_{\mathsf{c}}, \rho_{\mathsf{e}})$ if:

- ρ_{c} (constraint rate) is (at least) the maximum among the rates in $\vec{\rho} := (\vec{\rho}_i)_{i=1}^{\mathsf{k}}$ and the rates in $\{\sigma \,:\, \mathfrak{C} \in V\,,\, (\mathcal{C}, \sigma) \in \mathfrak{C}\}$;
- ρ_{e} (effective rate) is (at least) the maximum among ρ_{c} and the rates in $\{\mathrm{rate}(N; \vec{\rho}), \sigma + \mathrm{rate}(D) : \mathfrak{C} \in V\,,\, (\mathcal{C}, \sigma) \in \mathfrak{C}\}$.

Note that $\rho_{\mathsf{c}} \leq \rho_{\mathsf{e}}$. The definition of ρ_{e} may appear mysterious, but it is naturally motivated by the proof of Theorem 3.7.1.

Remark 2.3.7. The model of RS-encoded IOPs does not forbid the verifier from making queries to messages. However, in all of our protocols to achieve soundness it suffices for the rational constraints output by the verifier to be satisfied (and so the verifier does not make any queries). For this reason, we do not consider query complexity when discussing RS-encoded IOPs. Naturally, after we "compile" an RS-encoded IOP into a corresponding (regular) IOP, the resulting verifier will make queries to the proof; for details, see Section 3.7.

2.3.3.1 Proximity

In an RS-encoded IOP *of Proximity* (RS-encoded IOPP), soundness must hold only if prover messages are Reed–Solomon codewords *and* the witness is a tuple of Reed–Solomon codewords. Formally, a *Reed–Solomon IOPP system* for a relation $\mathcal{R} \subseteq \{0,1\}^n \times \mathrm{RS}\left[L, \vec{\rho}_{\mathtt{w}}\right]$ is a tuple $(P, V, (\vec{\rho}_i)_{i=1}^{\mathsf{k}})$, where P and V are probabilistic algorithms, that satisfies the properties below. Note that the rational constraints output by the verifier may now also take the witness as input; the definition of maximum rate is modified accordingly.

Completeness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ in the relation $\mathcal{R}$, $(P(\mathtt{x}, \mathtt{w}), V^{\mathtt{w}}(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol with accepting probability 1, where the i-th message of P is a codeword of $\mathrm{RS}\left[L, \vec{\rho}_i\right]$, and V outputs a set of rational constraints that are satisfied with respect to the witness and the prover's messages with probability 1.

Soundness: For every instance-witness pair $(\mathtt{x}, \mathtt{w})$ with $\mathtt{w} \in \left(\mathrm{RS}\left[L, \vec{\rho}_{\mathtt{w}}\right] \setminus \mathcal{R}|_{\mathtt{x}}\right)$ and unbounded malicious prover $\tilde{P}$ whose i-th message is a codeword of $\mathrm{RS}\left[L, \vec{\rho}_i\right]$, $(\tilde{P}, V^{\mathtt{w}}(\mathtt{x}))$ is a $\mathsf{k}(n)$-round interactive oracle protocol wherein the set of rational constraints output by V are satisfied with respect to the witness and the prover's messages with probability at most $\varepsilon(n)$.

While the soundness condition does not consider "distance" of candidate witnesses to $\mathcal{R}|_{\mathtt{x}}$ (as in Section 2.3.1.1), we think of the notion above as an IOPP because soundness holds with respect to a particular witness provided as an oracle to the verifier. (This is analogous to "exact" PCPPs in [105].)

2.3.3.2 Zero knowledge

The definition of zero knowledge for RS-encoded IOPs (resp., RS-encoded IOPPs) equals that for IOPs (resp., IOPPs). This is because the definitions of RS-encoded IOPs and (standard) IOPs differ only in the soundness condition. Note that while the honest verifiers that we consider never make queries, a malicious verifier may do so. Indeed, we *must* allow malicious verifiers to make queries in order to "lift" zero knowledge guarantees from an RS-encoded IOP to a corresponding (regular) IOP, and thereby achieve the notion of zero knowledge against a given query bound b stated in Section 2.3.2. We further note that the structure of the compiler that performs this lifting (see Section 3.7) motivates a definition of query bound b that can lead to more efficient constructions. Namely, since all of the prover messages and witnesses are over the same domain L, we merely count the number of *distinct* queries to this common domain, i.e., if a malicious verifier queries multiple prover messages (or witnesses) at the same position $\alpha \in L$, we consider it a single query.

2.3.4 Univariate rowcheck

We describe *univariate rowcheck*, a noninteractive RS-encoded IOPP for simultaneously testing satisfaction of a given arithmetic constraint on a large number of inputs. The next definition captures this.

Definition 2.3.8 (rowcheck relation)**.** *The relation $\mathcal{R}_{\mathrm{ROW}}$ is the set of all pairs*

$$\big((\mathbb{F}, L, H, \rho, \mathsf{w}, \mathsf{c}) , (f_1, \ldots, f_{\mathsf{w}}) \big)$$

where $\mathbb{F}$ is a finite field, L, H are affine subspaces of $\mathbb{F}$ with $L \cap H = \emptyset$, $\rho \in (0,1)$, $\mathsf{w} \in \mathbb{N}$, c is an arithmetic circuit, $f_1, \ldots, f_{\mathsf{w}} \in \mathrm{RS}\,[L, \rho]$, and $\forall\, a \in H$, $\mathsf{c}(\hat{f}_1(a), \ldots, \hat{f}_{\mathsf{w}}(a)) = 0$.

Standard techniques for testing membership in the *vanishing subcode* of the Reed–Solomon code [41] directly imply a non-interactive RS-encoded IOPP for the above rowcheck relation. Namely, the system of equations $\{\mathsf{c}(\hat{f}_1(a), \ldots, \hat{f}_{\mathsf{w}}(a)) = 0\}_{a \in H}$ is equivalent via the factor theorem to the statement "there exists $g \in \mathrm{RS}[L, \deg(\mathsf{c})\rho - |H|/|L|]$ such that $\hat{g}(X) \cdot \prod_{a \in H} (X - a)$ $\equiv \mathsf{c}(\hat{f}_1(X), \ldots, \hat{f}_{\mathsf{w}}(X))$". Therefore, the prover could send g to the verifier, who could probabilistically check the identity at a random point of L, with a soundness error of $\deg(\mathsf{c})\rho$. In fact, within the formalism of RS-encoded IOPPs (and given that $L \cap H = \emptyset$) there is no need for the prover to send anything: the verifier can simply check that $p \in \mathrm{RS}[L, \deg(\mathsf{c})\rho - |H|/|L|]$ for the function $p\colon L \to \mathbb{F}$ defined by

$$\forall\, a \in L\,,\ p(a) := \frac{\mathsf{c}(\hat{f}_1(a), \ldots, \hat{f}_{\mathsf{w}}(a))}{\mathbb{Z}_H(a)} \quad .$$

The maximum rate for the foregoing RS-encoded IOPP is $(\rho_{\mathsf{c}}, \rho_{\mathsf{e}}) = (\max\{\rho, \deg(\mathsf{c})\rho - |H|/|L|\}, \deg(\mathsf{c}) \cdot \rho)$. Note that the verifier can simulate oracle access to the function p when given oracle access to the witness oracles $f_1, \ldots, f_{\mathsf{w}}$. Each query to p requires evaluating the arithmetic circuit c and the vanishing polynomial $\mathbb{Z}_H$. Throughout, we directly use the above ideas without encapsulating them in "rowcheck sub-protocols".

Chapter 3

AURORA: an efficient IOP for R1CS

In this chapter we present AURORA, a zero knowledge SNARG of knowledge (zkSNARK) for (an extension of) arithmetic circuit satisfiability whose argument size is polylogarithmic in the circuit size. Aurora also has attractive features: it uses a transparent setup, is plausibly post-quantum secure, and only makes black-box use of fast symmetric cryptography (any cryptographic hash function modeled as a random oracle).

Our work makes an exponential asymptotic improvement in argument size over Ligero [9], a recent zero knowledge non-interactive argument with similar features but where proofs scale as the *square root* of the circuit size. For example, Aurora's proofs are $30\times$ smaller than Ligero's for circuits with a million gates (which already suffices for representative applications such as Zcash).

Our work also complements and improves on Stark [23], a recent zkSNARK that targets computations expressed as bounded halting problems on random access machines. While Stark was designed for a different computation model, we can still study its efficiency when applied to arithmetic circuits. In this case Aurora's prover is faster by a logarithmic factor (in the circuit size) and Aurora's proofs are concretely much shorter, e.g., $20\times$ smaller for circuits with a million gates.

The efficiency features of Aurora stem from a new Interactive Oracle Proof (IOP) that solves a *univariate* analogue of the important sumcheck problem [115], in which query complexity is *logarithmic* in the degree of the summand polynomial. This is an *exponential* improvement over the original multi-variate protocol, where communication complexity is (at least) *linear* in the degree of the polynomial. We believe this protocol and its analysis are of independent interest.

3.1 Contributions of this chapter

We present several contributions: (1) an IOP protocol for R1CS with attractive efficiency features; (2) design, implementation, and evaluation of a transparent zkSNARK for R1CS, based on this IOP; (3) a library for writing IOP-based non-interactive arguments. We now describe each contribution.

(1) IOP for R1CS. We construct a zero knowledge IOP protocol for rank-1 constraint satisfaction (R1CS) with *linear* proof length and *logarithmic* query complexity.

Given an R1CS instance $\mathcal{C} = (A, B, C)$ with $A, B, C \in \mathbb{F}^{m \times n}$, we denote by $N = \Omega(m+n)$ the total number of non-zero entries in the three matrices and by $|\mathcal{C}|$ the number of bits required to represent these; note that $|\mathcal{C}| = \Theta(N \log |\mathbb{F}|)$. One can view N as the number of "arithmetic gates" in the R1CS instance.

Theorem 3.1.1 (informal)**.** *There is an $O(\log N)$-round IOP protocol for R1CS with proof length $O(N)$ over alphabet $\mathbb{F}$ and query complexity $O(\log N)$. The prover uses $O(N \log N)$ field operations, while the verifier uses $O(N)$ field operations. The IOP protocol is public coin and is a zero knowledge proof of knowledge.*

The core of our result is a solution to a *univariate* analogue of the classical sumcheck problem [115]. Our protocol (including zero knowledge and soundness error reduction) is relatively simple: it is specified in a single page (see Fig. 3.3 in Section 3.8), given a low-degree test as a subroutine. The low degree test that we use is a recent highly-efficient IOP for testing proximity to the Reed–Solomon code [22].

(2) zkSNARK for R1CS. We design, implement, and evaluate **Aurora**, a zero knowledge SNARG of knowledge (zkSNARK) for R1CS with several notable features: (a) it only makes black-box use of fast symmetric cryptography (any cryptographic hash function modeled as a random oracle); (b) it has a transparent setup (users merely need to "agree" on which cryptographic hash function to use); (c) it is plausibly post-quantum secure (there are no known efficient quantum attacks against this construction). These features follow from the fact that Aurora is obtained by applying the transformation of [32] to our IOP for R1CS. This transformation preserves both zero knowledge and proof of knowledge of the underlying IOP. The following theorem is obtained straightforwardly by combining Theorem 3.1.1 with [32, Theorem 7.1].

Theorem 3.1.2 (informal)**.** *There exists a zkSNARK for R1CS that is unconditionally secure in the random oracle model with proof length $O_\lambda(\log^2 N)$. The prover runs in time $O_\lambda(N \log N)$ and the verifier in time $O_\lambda(N)$. (Here for simplicity we take the field $\mathbb{F}$ to have size $2^{\Theta(\lambda)}$ where λ is the security parameter.)*

For example, setting our implementation to a security level of 128 bits over a 192-bit finite field, proofs range from $40\,\mathrm{kB}$ to $130\,\mathrm{kB}$ for instances of up to millions of gates; producing proofs takes on the order of several minutes and checking proofs on the order of several seconds. (See Section 5.13 for details.)

Overall, as indicated in Fig. 1.2, we achieve the smallest argument size among (plausibly) post-quantum non-interactive arguments for circuits, *by more than an order of magnitude*. Other approaches achieve smaller argument sizes by relying on (public-key) cryptography that is insecure against quantum adversaries.

(3) `libiop`: a library for non-interactive arguments. We provide `libiop`, a codebase that enables the design and implementation of non-interactive arguments based on IOPs. The codebase uses the C++ language and has three main components: (1) a library for writing IOP protocols; (2) a realization of [32]'s transformation, mapping any IOP written with our library to a corresponding non-interactive argument; (3) a portfolio of IOP protocols. We have released `libiop` under a permissive software license for the community (see `https://github.com/scipr-lab/libiop`). We believe that our library will serve as a useful tool in meeting the increasing demand by practitioners for transparent non-interactive arguments.

3.2 Techniques

Our main technical contribution is a linear-length logarithmic-query IOP for R1CS (Theorem 3.1.1), which we use to design, implement, and evaluate a transparent zkSNARK for R1CS. Below we summarize the main ideas behind our protocol, and postpone to Sections 3.9 and 5.13 discussions of our system. In Section 3.2.1, we describe our approach to obtain the IOP for R1CS; this approach leads us to solve the univariate sumcheck problem, as discussed in Section 3.2.2; finally, in Section 3.2.3, we explain how we achieve zero knowledge. In Section 3.2.4 we conclude with a wider perspective on the techniques used in this paper.

3.2.1 Our interactive oracle proof for R1CS

The R1CS relation consists of instance-witness pairs $((A, B, C, v), w)$, where A, B, C are matrices and v, w are vectors over a finite field $\mathbb{F}$, such that $(Az) \circ (Bz) = Cz$ for $z := (1, v, w)$ and "$\circ$" denotes the entry-wise product.[1] For example, R1CS captures arithmetic circuit satisfaction: A, B, C represent the circuit's gates, v the circuit's public input, and w the circuit's private input and wire values.[2]

We describe the high-level structure of our IOP protocol for R1CS, which has linear proof length and logarithmic query complexity. The protocol tests satisfaction by relying on two building blocks, one for testing the entry-wise vector product and the other for testing the linear transformations induced by the matrices A, B, C. Informally, we thus consider protocols for the following two problems.

- **Rowcheck:** given vectors $x, y, z \in \mathbb{F}^m$, test whether $x \circ y = z$, where "$\circ$" denotes entry-wise product.
- **Lincheck:** given vectors $x \in \mathbb{F}^m, y \in \mathbb{F}^n$ and a matrix $M \in \mathbb{F}^{m \times n}$, test whether $x = My$.

One can immediately obtain an IOP for R1CS when given IOPs for the rowcheck and lincheck problems. The prover first sends four oracles to the verifier: the satisfying assignment z and its linear transformations $y_A := Az, y_B := Bz, y_C := Cz$. Then the prover and verifier engage in four IOPs in parallel:
– An IOP for the lincheck problem to check that "$y_A = Az$". Likewise for y_B and y_C.
– An IOP for the rowcheck problem to check that "$y_A \circ y_B = y_C$".
Finally, the verifier checks that z is consistent with the public input v. Clearly, there exist z, y_A, y_B, y_C that yield valid rowcheck and lincheck instances if and only if (A, B, C, v) is a satisfiable R1CS instance.

[1] Throughout, we assume that $\mathbb{F}$ is "friendly" to FFT algorithms, i.e., $\mathbb{F}$ is a binary field or its multiplicative group is smooth.

[2] The reader may be familiar with a standard arithmetization of circuit satisfaction (used, e.g., in the inner PCP of [10]). Given an arithmetic circuit with m gates and n wires, each addition gate $x_i \leftarrow x_j + x_k$ is mapped to the linear constraint $x_i = x_j + x_k$ and each product gate $x_i \leftarrow x_j \cdot x_k$ is mapped to the quadratic constraint $x_i = x_j \cdot x_k$. The resulting system of equations can be written as $A \cdot ((1, x) \otimes (1, x)) = b$ for suitable $A \in \mathbb{F}^{m \times (n+1)^2}$ and $b \in \mathbb{F}^m$. However, this reduction results in a quadratic blowup in the instance size. There is an alternative reduction due to [117, 87] that avoids this.

The foregoing reduces the goal to designing IOPs for the rowcheck and lincheck problems.

As stated, however, the rowcheck and lincheck problems only admit "trivial" protocols in which the verifier queries all entries of the vectors in order to check the required properties. In order to allow for sublinear query complexity, we need the vectors x, y, z to be *encoded* via some error-correcting code. We use the Reed–Solomon (RS) code because it ensures constant distance with constant rate while at the same time it enjoys efficient IOPs of Proximity [22].

Given an evaluation domain $L \subseteq \mathbb{F}$ and rate parameter $\rho \in [0, 1]$, $\mathrm{RS}[L, \rho]$ is the set of all codewords $f \colon L \to \mathbb{F}$ that are evaluations of polynomials of degree less than $\rho|L|$. Then, the encoding of a vector $v \in \mathbb{F}^S$ with $S \subseteq \mathbb{F}$ and $|S| < \rho|L|$ is $\hat{v}|_L \in \mathbb{F}^L$ where $\hat{v}$ is the unique polynomial of degree $|S| - 1$ such that $\hat{v}|_S = v$. Given this encoding, we consider "encoded" variants of the rowcheck and lincheck problems.

- **Univariate rowcheck** (Definition 2.3.8): given a subset $H \subseteq \mathbb{F}$ and codewords $f, g, h \in \mathrm{RS}[L, \rho]$, check that $\hat{f}(a) \cdot \hat{g}(a) - \hat{h}(a) = 0$ for all $a \in H$. (This is a special case of the definition that we use later.)

- **Univariate lincheck** (Definition 3.5.1): given subsets $H_1, H_2 \subseteq \mathbb{F}$, codewords $f, g \in \mathrm{RS}[L, \rho]$, and a matrix $M \in \mathbb{F}^{H_1 \times H_2}$, check that $\hat{f}(a) = \sum_{b \in H_2} M_{a,b} \cdot \hat{g}(b)$ for all $a \in H_1$.

Given IOPs for the above problems, we can now get an IOP protocol for R1CS roughly as before. Rather than sending z, Az, Bz, Cz, the prover sends their encodings $f_z, f_{Az}, f_{Bz}, f_{Cz}$. The prover and verifier then engage in rowcheck and lincheck protocols as before, but with respect to these encodings.

For these encoded variants, we achieve IOP protocols with linear proof length and logarithmic query complexity, as required. We obtain a protocol for rowcheck via standard techniques from the probabilistic checking literature [41]. As for lincheck, we do not use any routing and instead use a technique (dating back at least to [16]) to reduce the given testing problem to a *sumcheck instance*. However, since we are not working with multivariate polynomials, we cannot rely on the usual (multivariate) sumcheck protocol. Instead, we present a novel protocol that realizes a univariate analogue of the classical sumcheck protocol, and use it as the testing "core" of our IOP protocol for R1CS. We discuss univariate sumcheck next.

Remark 3.2.1. The verifier receives as input an explicit (non-uniform) description of the set of constraints, namely, the matrices A, B, C. In particular, the verifier runs in time that is at least linear in the number of non-zero entries in these matrices (if we consider a sparse-matrix representation for example).

3.2.2 A sumcheck protocol for univariate polynomials

A key ingredient in our IOP protocol is a *univariate* analogue of the classical (multivariate) sumcheck protocol [115]. Recall that the classical sumcheck protocol is an IP for claims of the form "$\sum_{\vec{a} \in H^m} f(\vec{a}) = 0$", where f is a given polynomial in $\mathbb{F}[X_1, \ldots, X_m]$ of individual degree d and H is a subset of $\mathbb{F}$. In this protocol, the verifier runs in time $\mathrm{poly}(m, d, \log |\mathbb{F}|)$ and accesses f at a single (random) location. The sumcheck protocol plays a fundamental role in computational

complexity (it underlies celebrated results such as $\mathsf{IP} = \mathsf{PSPACE}$ [137] and $\mathsf{MIP} = \mathsf{NEXP}$ [17]) and in efficient proof protocols [90, 73, 143, 141, 142, 145, 146, 153, 154, 148].

We work with univariate polynomials instead, and need a univariate analogue of the sumcheck protocol (see previous subsection): *how can a prover convince the verifier that "*$\sum_{a \in H} f(a) = 0$*" for a given polynomial* $f \in \mathbb{F}[X]$ *of degree* d *and subset* $H \subseteq \mathbb{F}$*?* Designing a "univariate sumcheck" is not straightforward because univariate polynomials (the Reed–Solomon code) do not have the tensor structure used by the sumcheck protocol for multivariate polynomials (the Reed–Muller code). In particular, the sumcheck protocol has m rounds, each of which reduces a sumcheck problem to a simpler sumcheck problem with one variable fewer. When there is only one variable, however, it is not clear to what simpler problems one can reduce.

Using different ideas, we design a natural protocol for univariate sumcheck in the cases where H is an additive or multiplicative coset in $\mathbb{F}$ (i.e., a coset of an additive or multiplicative subgroup of $\mathbb{F}$).

Theorem (informal)**.** *The univariate sumcheck protocol over additive or multiplicative cosets has a* $O(\log d)$*-round IOP with proof complexity* $O(d)$ *over alphabet* $\mathbb{F}$ *and query complexity* $O(\log d)$*. The IOP prover uses* $O(d \log |H|)$ *field operations and the IOP verifier uses* $O(\log d + \log^2 |H|)$ *field operations.*

We now provide the main ideas behind the protocol, when H is an *additive* coset in $\mathbb{F}$.

Suppose for a moment that the degree d of f is less than $|H|$ (we remove this restriction later). A theorem of Byott and Chapman [59] states that the sum of f over (an additive coset) H is zero if and only if the coefficient of $X^{|H|-1}$ in f is zero. In particular, $\sum_{a \in H} f(a)$ is zero if and only if f has degree less than $|H| - 1$. Thus, the univariate sumcheck problem over H when $d < |H|$ is equivalent to low-degree testing.

The foregoing suggests a natural approach: test that f has degree less than $|H| - 1$. Without any help from the prover, the verifier would need at least $|H|$ queries to f to conduct such a test, which is as expensive as querying all of H. However, the prover can help by engaging with the verifier in an IOP of Proximity for the Reed–Solomon code. For this we rely on the recent construction of Ben-Sasson et al. [22], which has proof length $O(d)$ and query complexity $O(\log d)$.

In our setting, however, we need to also handle the case where the degree d of f is larger than $|H|$. For this case, we observe that we can split any polynomial f into two polynomials g and h such that $f(x) \equiv g(x) + \prod_{\alpha \in H}(x - \alpha) \cdot h(x)$ with $\deg(g) < |H|$ and $\deg(h) < d - |H|$; in particular, f and g agree on H, and thus so do their sums on H. This observation suggests the following extension to the prior approach: the prover sends g (as an oracle) to the verifier, and then the verifier performs the prior protocol with g in place of f. Of course, a cheating prover may send a polynomial g that has nothing to do with f, and so the verifier must also ensure that g is consistent with f. To facilitate this, we actually have the prover send h rather than g; the verifier can then "query" $g(x)$ as $f(x) - \prod_{\alpha \in H}(x - \alpha) \cdot h(x)$; the prover then shows that f, g, h are all of the correct degrees.

A similar reasoning works when H is a multiplicative coset in $\mathbb{F}$ (see Remark 3.4.6). It remains an interesting open problem to establish whether the foregoing can be extended to any subset H in $\mathbb{F}$.

Remark 3.2.2 (vanishing vs. summing)**.** The following are both linear subcodes of the Reed–Solomon code:

$$\mathsf{VanishRS}[\mathbb{F}, L, H, d] := \{f \colon L \to \mathbb{F} \mid f \text{ has degree} < d \text{ and is zero everywhere on } H\} \ ,$$
$$\mathsf{SumRS}[\mathbb{F}, L, H, d] := \{f \colon L \to \mathbb{F} \mid f \text{ has degree} < d \text{ and sums to zero on } H\} \ .$$

Our univariate sumcheck protocol is an IOP of Proximity for SumRS, and is reminiscent of IOPs of Proximity for VanishRS (e.g., see [23]). Nevertheless, there are also intriguing differences between the two cases. For example, while it is known how to test proximity to VanishRS for general H, we only know how to test proximity to SumRS when H is a coset. Additionally, our IOP protocol for R1CS from Section 3.2.1 can be viewed as a reduction from checking satisfaction of R1CS to testing proximity to SumRS; we do not know how to carry out a similar reduction to VanishRS. Indeed, there is an interactive reduction from VanishRS to SumRS, but no reduction in the other direction is known.

3.2.3 Efficient zero knowledge from algebraic techniques

The ideas discussed thus far yield an IOP protocol for R1CS with linear proof length and logarithmic query complexity. However these by themselves do not provide zero knowledge.

We achieve zero knowledge by leveraging recent algebraic techniques [26]. Informally, we adapt these techniques to achieve efficient zero knowledge variants of key sub-protocols, including the univariate sumcheck protocol (see Section 3.4.1) and low-degree testing (see Section 3.7.1), and combine these to achieve a zero knowledge IOP protocol for R1CS (see Sections 3.6.1 and 3.8).

We summarize the basic intuition for how we achieve zero knowledge in our protocols.

First, we use *bounded independence*. Informally, rather than encoding a vector $z \in \mathbb{F}^H$ by the unique polynomial of degree $|H| - 1$ that matches z on H, we instead sample uniformly at random a polynomial of degree, say, $|H| + 9$ conditioned on matching z on H. Any set of 10 evaluations of such a polynomial are independently and uniformly distributed in $\mathbb{F}$ (and thus reveal no information about z), *provided these evaluations are outside of* H. To ensure this latter condition, we choose the evaluation domain L of Reed–Solomon codewords to be disjoint from H. Thus, for example, if H is a linear space (an additive subgroup of $\mathbb{F}$) then we choose L to be an affine subspace (a coset of some additive subgroup), since the underlying machinery for low-degree testing (e.g., [22]) requires codewords to be evaluated over algebraically-structured domains. All of our protocols are robust to these variations.

Bounded independence alone does not suffice, though. For example, in the sumcheck protocol, consider the case where the input vector $z \in \mathbb{F}^H$ is all zeroes. The prover samples a random polynomial $\hat{f}$ of degree $|H| + 9$, such that $\hat{f}(a) = 0$ for all $a \in H$, and sends its evaluation f over L disjoint from H to the verifier. As discussed, any ten queries to f result in ten independent and uniformly random elements in $\mathbb{F}$. Observe, however, that when we run the sumcheck protocol on f, the polynomial g (the remainder of $\hat{f}$ when divided by $\prod_{\alpha \in H}(x - \alpha)$) is the zero polynomial: all randomness is removed by the division.

To remedy this, we use *self-reducibility* to reduce a sumcheck claim about the polynomial f to a sumcheck claim about a random polynomial. The prover first sends a random Reed–Solomon

codeword r, along with the value $\beta := \sum_{a \in H} r(a)$. The verifier sends a random challenge $\rho \in \mathbb{F}$. Then the prover and verifier engage in the univariate sumcheck protocol with respect to the new claim "$\sum_{a \in H} \rho f(a) + r(a) = \beta$". Since r is uniformly random, $\rho f + r$ is uniformly random for any ρ, and thus the sumcheck protocol is performed on a random polynomial, which ensures zero knowledge. Soundness is ensured by the fact that if f does not sum to 0 on H then the new claim is true with probability $1/|\mathbb{F}|$ over the choice of ρ.

3.2.4 Perspective on our techniques

A linear-length logarithmic-query IOP for a "circuit-like" NP-complete relation like R1CS (Theorem 3.1.1) may come as a surprise. We wish to shed some light on our IOP construction by connecting the ideas behind it to prior ideas in the probabilistic checking literature, and use these connections to motivate our construction.

A significant cost in all known PCP constructions with good proof length is using *routing networks* to reduce combinatorial objects (circuits, machines, and so on) to structured algebraic ones;[3] routing also plays a major role in many IOPs [26, 24, 21, 23]. While it is plausible that one could adapt routing techniques to route the constraints of an R1CS instance (similarly to [126]), such an approach would likely incur logarithmic-factor overheads, precluding *linear*-size IOPs.

A recent work [25] achieves linear-length constant-query IOPs for boolean circuit satisfaction *without routing the input circuit*. Unfortunately, [25] relies on other expensive tools, such as algebraic-geometry (AG) codes and quasilinear-size PCPs of proximity [41]; moreover, it is not zero knowledge. Informally, [25] tests arbitrary (unstructured) constraints by invoking a sumcheck protocol [115] on a $O(1)$-wise tensor product of AG codes; this latter is then locally tested via tools in [40, 41].

One may conjecture that, to achieve an IOP for R1CS like ours, it would suffice to merely replace the AG codes in [25] with the Reed–Solomon code, since both codes have constant rate. But taking a tensor product exponentially deteriorates rate, and testing proximity to that tensor product would be expensive.

An alternative approach is to solve a sumcheck problem *directly* on the Reed–Solomon code. Existing protocols are not of much use here: the multivariate sumcheck protocol relies on a tensor structure that is *not* available in the Reed–Solomon code, and recent IOP implementations either use routing [21, 23] or achieve only sublinear query complexity [9].

Instead, we design a completely new IOP for a sumcheck problem on the Reed–Solomon code. We then combine this solution with ideas from [25] (to avoid routing) and from [26] (to achieve zero knowledge) to obtain our linear-length logarithmic-query IOP for R1CS. Along the way, we rely on recent efficient proximity tests for the Reed–Solomon code [22].

[3] Polishchuk and Spielman [126] reduce boolean circuit satisfaction to a trivariate algebraic coloring problem with "low-degree" neighbor relations, by routing the circuit's wires over an arithmetized routing network. Ben-Sasson and Sudan [41] reduce nondeterministic machine computations to a univariate algebraic satisfaction problem by routing the machine's memory accesses over another arithmetized routing network. Routing is again a crucial component in the linear-size sublinear-query PCPs of [38].

3.3 Roadmap

Subsequent sections describe subprotocols, presented as *Reed–Solomon encoded* IOPs, which are IOPs for which soundness only holds against provers whose messages are Reed–Solomon codewords of specified rates, that are later compiled into standard IOPs. In Section 3.6 we combine the rowcheck and lincheck protocols to obtain an RS-encoded IOP for R1CS. In Section 3.7 we explain how to transform RS-encoded IOPs to standard IOPs, and in Section 3.8 we apply this transformation to our RS-encoded IOP for R1CS. Fig. 3.1 summarizes the structure of our IOP for R1CS. Finally, in Section 3.9 we describe our implementation and in Section 5.13 we report on its evaluation.

Throughout, we focus on the case where all relevant domains are *additive* cosets (affine subspaces) in $\mathbb{F}$. The case where domains are *multiplicative* cosets is similar, with only minor modifications (see Remark 3.4.6). Moreover, while for convenience we limit our discussions to establishing soundness, all protocols described in this paper are easily seen to satisfy the stronger notion of proof of knowledge. Informally, this is because we prove soundness by showing that oracles sent by convincing provers can be decoded to valid witnesses.

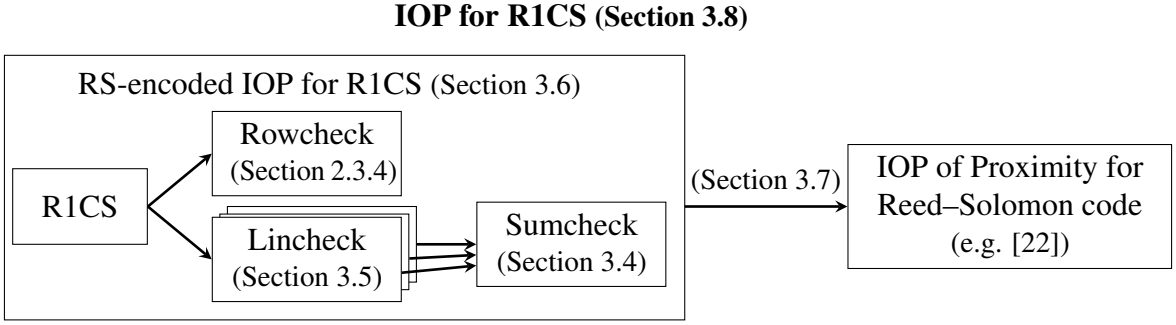


Figure 3.1: Structure of our IOP for R1CS in terms of key sub-protocols.

3.4 Univariate sumcheck

We describe *univariate sumcheck*, an RS-encoded IOPP for testing whether a low-degree univariate polynomial $\hat{f}$ sums to zero on a given subspace $H \subseteq \mathbb{F}$. This protocol is a univariate analogue of the multi-variate sumcheck protocol [115].

If $\hat{f}$ has degree less than d, then $\hat{f}$ can be uniquely decomposed into polynomials $\hat{g}, \hat{h}$ of degrees less than $|H|$ and $d-|H|$ (respectively) such that $\hat{f} \equiv \hat{g} + \mathbb{Z}_H \cdot \hat{h}$, where $\mathbb{Z}_H$ is the vanishing polynomial of H (see Section 2.2.3). This implies that $\sum_{a \in H} \hat{f}(a) = \sum_{a \in H} (\hat{g}(a) + \mathbb{Z}_H(a) \cdot \hat{h}(a)) = \sum_{a \in H} \hat{g}(a)$. By Lemma 3.4.4 below, this latter expression is equal to $\beta \sum_{a \in H} a^{|H|-1}$, where β is the coefficient of $X^{|H|-1}$ in $\hat{g}(X)$. Note that $\sum_{a \in H} a^{|H|-1} \neq 0$ since otherwise this would imply that all functions sum to zero on H. Thus, $\sum_{a \in H} \hat{f}(a) = 0$ if and only if $\beta = 0$.

This suggests the following RS-encoded IOPP (actually an RS-encoded PCPP). The prover sends g, h (the evaluations of $\hat{g}, \hat{h}$). The verifier now must check that (a) $\hat{f} \equiv \hat{g} + \mathbb{Z}_H \cdot \hat{h}$, and (b) the coefficient of $X^{|H|-1}$ in $\hat{g}$ is zero. For both conditions we use the definition of

an RS-encoded IOPP: the verifier outputs a rational constraint specifying that the polynomial $\hat{f} - \mathbb{Z}_H \cdot \hat{h}$ is of degree less than $|H| - 1$, which corresponds to forcing the coefficient of $X^{|H|-1}$ to be zero. In the final (non-encoded) IOPP protocol this will correspond to testing proximity of $\hat{f} - \mathbb{Z}_H \cdot \hat{h}$ to a Reed–Solomon code with rate parameter $(|H| - 1)/|L|$.

Below we consider the more general case of testing that the sum equals a given $\mu \in \mathbb{F}$ (rather than zero).

Definition 3.4.1 (sumcheck relation)**.** *$\mathcal{R}_{\text{SUM}}$ is the set of pairs $\left((\mathbb{F}, L, H, \rho, \mu), f \right)$ where $\mathbb{F}$ is a finite field, L, H are affine subspaces of $\mathbb{F}$, $\rho \in (0, 1)$, $\mu \in \mathbb{F}$, $f \in \text{RS}[L, \rho]$, and $\sum_{a \in H} \hat{f}(a) = \mu$.*

Theorem 3.4.2. *There exists an RS-encoded IOPP (Protocol 3.4.3) for the sumcheck relation $\mathcal{R}_{\text{SUM}}$ with the following parameters:*

alphabet	Σ	$=$	$\mathbb{F}$
number of rounds	k	$=$	1
proof length	p	$=$	$\lvert L \rvert$
randomness	r	$=$	0
soundness error	ε	$=$	0
prover time	t_{P}	$=$	$O(\lvert L \rvert \log \lvert H \rvert) + 2 \cdot \text{FFT}(\mathbb{F}, \lvert L \rvert)$
verifier time	t_{V}	$=$	$O(\log^2 \lvert H \rvert)$
maximum rate	$(\rho_{\mathsf{c}}, \rho_{\mathsf{e}})$	$=$	(ρ, ρ)

Protocol 3.4.3. Let $f \in \text{RS}[L, \rho]$ be the witness oracle, and let $\hat{f}$ be the unique polynomial of degree at most $\rho|L|$ that agrees with f. The RS-encoded IOP protocol (P, V) for $\mathcal{R}_{\text{SUM}}$ proceeds as follows.

1. P computes the unique polynomials $\hat{g}$ and $\hat{h}$ and unique element $\beta \in \mathbb{F}$ such that $\deg(\hat{g}) < |H| - 1$, $\deg(\hat{h}) < \rho|L| - |H|$, and $\hat{f}(X) \equiv \hat{g}(X) + \beta X^{|H|-1} + \mathbb{Z}_H(X)\hat{h}(X)$.
2. P sends $h := \hat{h}|_L \in \text{RS}[L, \rho - |H|/|L|]$ to V.
3. V computes $\xi := \sum_{a \in H} a^{|H|-1}$ (this can be done efficiently as explained below), and accepts if and only if $p \in \text{RS}[L, (|H|-1)/|L|]$ where $\hat{p}(X) := \xi \cdot \hat{f}(X) - \mu \cdot X^{|H|-1} - \xi \cdot \mathbb{Z}_H(X)\hat{h}(X)$. In the formalism of RS-encoded IOPs (see Section 2.3.3), this corresponds to the rational constraint $(\mathcal{C}, \sigma) := ((N, D), \frac{|H|-1}{|L|})$ where $N(X, Z_1, Z_2) := \xi \cdot Z_1 - \mu \cdot X^{|H|-1} - \xi \cdot \mathbb{Z}_H(X) \cdot Z_2$ and $D(X) := 1$.

Note that the maximum rate in the above protocol (as defined in Section 2.3.3) is ρ.

Proof. Completeness and soundness rely on the following lemma:

Lemma 3.4.4 ([59, Theorem 1], restated)**.** *Let H be an affine subspace of $\mathbb{F}$, and let $\hat{g}(x)$ be a univariate polynomial over $\mathbb{F}$ of degree (strictly) less than $|H| - 1$. Then*

$$\sum_{a \in H} \hat{g}(a) = 0.$$

We provide a self-contained proof of this statement in Appendix A.1.

Completeness. Consider $f \in \mathrm{RS}\,[L, \rho]$ with $\sum_{a \in H} \hat{f}(a) = \mu$. Then, by definition of g, h and Lemma 3.4.4,

$$\mu = \sum_{a \in H} \Big(\hat{g}(a) + \beta \cdot a^{|H|-1} + \mathbb{Z}_H(a)\hat{h}(a) \Big) = \beta\xi \ .$$

Therefore,

$$\begin{aligned} &\xi \cdot \hat{f}(X) - \mu \cdot X^{|H|-1} - \xi \cdot \mathbb{Z}_H(X)\hat{h}(X) \\ &\equiv \xi \cdot \big(\hat{g}(X) + \beta X^{|H|-1} + \mathbb{Z}_H(X)\hat{h}(X) \big) - \mu \cdot X^{|H|-1} - \xi \cdot \mathbb{Z}_H(X)\hat{h}(X) \\ &\equiv \xi \cdot \hat{g}(X) + \xi\beta X^{|H|-1} - \mu \cdot X^{|H|-1} \equiv \xi \cdot \hat{g}(X) \ . \end{aligned}$$

Hence $\hat{p}(X) \equiv \xi \cdot \hat{g}(X)$, and so $p \in \mathrm{RS}[L, \frac{|H|-1}{|L|}]$.

Soundness. Consider $f \in \mathrm{RS}\,[L, \rho]$ with $\sum_{a \in H} \hat{f}(a) = \mu' \neq \mu$. We show that for any $h \in \mathrm{RS}[L, \rho - \frac{|H|}{|L|}]$, it holds that $p \notin \mathrm{RS}[L, \frac{|H|-1}{|L|}]$. Suppose towards contradiction that $p \in \mathrm{RS}[L, \frac{|H|-1}{|L|}]$. Then, by Lemma 3.4.4, we have that $\sum_{a \in H} \hat{p}(a) = 0$. But also we have that

$$\sum_{a \in H} \hat{p}(a) = \sum_{a \in H} (\xi \cdot \hat{f}(a) - \mu \cdot a^{|H|-1}) = \xi(\mu' - \mu) \neq 0$$

since $\xi \neq 0$. This is a contradiction.

Efficiency. For computational efficiency of the verifier, we use an additional lemma due to [59].

Lemma 3.4.5 ([59], implicit in the proof of Theorem 1)**.** *If H is an affine subspace of $\mathbb{F}$, then $\sum_{a \in H} a^{|H|-1}$ equals the linear term of $\mathbb{Z}_H$.*

The verifier runs in time $O(\log^2 |H|)$: its work consists of finding the linear term of $\mathbb{Z}_H$, which can be achieved via a divide-and-conquer algorithm, and evaluating $\mathbb{Z}_H$ at a single point. The prover runs in time $O(|L| \log |H|) + 2 \cdot \mathrm{FFT}(\mathbb{F}, |L|)$: the polynomial division can be performed by interpolating (one IFFT) over L to obtain the coefficients of f, running a divide-and-conquer algorithm to obtain the $O(\log |H|)$ coefficients of $\mathbb{Z}_H$, performing standard polynomial division to obtain $\hat{h}$, and computing its evaluation h via an FFT. □

Remark 3.4.6. The univariate sumcheck relation states that H, L are affine subspaces of $\mathbb{F}$ (Definition 3.4.1). One can define a similar relation where H, L are *multiplicative* cosets in $\mathbb{F}$, in which case Theorem 3.4.2 holds essentially unchanged. The protocol is similar to Protocol 3.4.3, except that $\hat{g}$ and $\hat{h}$ are such that $\hat{f}(X) = X \cdot \hat{g}(X) + \beta + \mathbb{Z}_H(X)\hat{h}(X)$. The rational constraint becomes $(\mathcal{C}, \sigma) := ((N, D), \frac{|H|-1}{|L|})$ where $N(X, Z_1, Z_2) := |H| \cdot Z_1 - \mu - |H| \cdot \mathbb{Z}_H(X) \cdot Z_2$, $D(X) := X$. Correctness of this protocol follows from the fact that, if H is a multiplicative coset, $\sum_{\alpha \in H} \hat{p}(\alpha) = \hat{p}(0) \cdot |H|$ for all polynomials $\hat{p}$ with $\deg(\hat{p}) < |H|$.

3.4.1 Zero knowledge

We describe how to modify Protocol 3.4.3 to achieve zero knowledge; the modification is an adaptation of algebraic techniques from [26, 24]. The prover first sends a random Reed–Solomon codeword $q \in \mathrm{RS}[L, \rho]$. The verifier then replies with a random "challenge" element $c \in \mathbb{F}$. Finally, the prover and verifier engage in Protocol 3.4.3 with respect to the "virtual" oracle $p := c \cdot f + q$, and new target value $c \cdot \mu + \sum_{a \in H} \hat{q}(a)$. Since p is an (almost) uniformly random Reed–Solomon codeword, one can efficiently simulate the sumcheck prover with input p. We obtain the following theorem.

Theorem 3.4.7. *There exists an RS-encoded IOPP (Protocol 3.4.8) for the sumcheck relation $\mathcal{R}_{\mathrm{SUM}}$ (Definition 3.4.1), which is zero knowledge against unbounded queries, with the following parameters:*

alphabet	Σ	$=$	$\mathbb{F}$
number of rounds	k	$=$	1
proof length	p	$=$	$2\lvert L\rvert$
randomness	r	$=$	$\log \lvert\mathbb{F}\rvert$
soundness error	ε	$=$	$1/\lvert\mathbb{F}\rvert$
prover time	$\mathsf{t_P}$	$=$	$O(\lvert L\rvert \log \lvert H\rvert) + 3 \cdot \mathrm{FFT}(\mathbb{F}, \lvert L\rvert)$
verifier time	$\mathsf{t_V}$	$=$	$O(\log^2 \lvert H\rvert)$
maximum rate	$(\rho_{\mathsf{c}}, \rho_{\mathsf{e}})$	$=$	(ρ, ρ)

Protocol 3.4.8. Let $f \in \mathrm{RS}[L, \rho]$ be the witness oracle. Let $(P_{\mathrm{SUM}}, V_{\mathrm{SUM}})$ be the RS-encoded IOP for $\mathcal{R}_{\mathrm{SUM}}$ (Protocol 3.4.3). The zero knowledge RS-encoded IOP (P, V) for $\mathcal{R}_{\mathrm{SUM}}$ proceeds as follows.

1. P samples $q \in \mathrm{RS}[L, \rho]$ uniformly at random and sends it to V, along with $\beta := \sum_{a \in H} q(a)$.
2. V samples $c \in \mathbb{F}$ uniformly at random, and sends it to P.
3. P and V invoke $\left(P_{\mathrm{SUM}}(\mathbb{x}', c \cdot f + q), V_{\mathrm{SUM}}^{c \cdot f + q}(\mathbb{x}')\right)$, where $\mathbb{x}' := (\mathbb{F}, L, H, \rho, c \cdot \mu + \beta)$.

Proof.

Completeness. Follows from the completeness of univariate sumcheck.

Soundness. Suppose that $\sum_{a \in H} \hat{f}(a) = \alpha \neq \mu$. Let $\beta' := \sum_{a \in H} q'(a)$, where q' is sent by $\tilde{P}$ in the first round. Then $\sum_{a \in H}(c \cdot \hat{f} + q')(a) = c \cdot \alpha + \beta'$, which is equal to $c \cdot \mu + \beta$ if and only if $c = \frac{\beta - \beta'}{\alpha - \mu}$, which happens with probability $1/|\mathbb{F}|$ for any fixed β, β'. Hence with probability $1 - 1/|\mathbb{F}|$, $(\mathbb{x}', c \cdot \mu + \beta) \notin \mathcal{R}_{\mathrm{SUM}}$, and soundness follows by the soundness of the standard protocol.

Zero knowledge. We describe a simulator S that, given straightline access to a (malicious) verifier $\tilde{V}$ and oracle access to a witness oracle $f \in \mathrm{RS}[L, \rho]$, perfectly simulates $\tilde{V}$'s view in the real protocol.

1. Sample $q_{\mathrm{sim}} \in \mathrm{RS}[L, \rho]$ uniformly at random and start simulating $\tilde{V}$.
2. Answer any query to f by querying f, and answer any query to q by querying q_{sim}. Let $Q_{\mathrm{sim}} \subseteq L$ be $\tilde{V}$'s queries to q from the beginning of the simulation until the next step.
3. Send $\beta_{\mathrm{sim}} := \sum_{a \in H} \hat{q}_{\mathrm{sim}}(a)$ to $\tilde{V}$.

4. Receive $\tilde{c}_{\text{sim}} \in \mathbb{F}$ from $\tilde{V}$.
5. Sample $p_{\text{sim}} \in \text{RS}[L, \rho]$ uniformly at random such that, for every $\omega \in Q_{\text{sim}}$, $p_{\text{sim}}(\omega) = \tilde{c}_{\text{sim}} \cdot f(\omega) + q_{\text{sim}}(\omega)$ and $\sum_{a \in H} p_{\text{sim}}(a) = \tilde{c}_{\text{sim}} \cdot \mu + \beta_{\text{sim}}$; this requires $|Q_{\text{sim}}|$ queries to f. (Note that if $|Q_{\text{sim}}| > \rho|L|$ then $p_{\text{sim}} \equiv \hat{f} + r_{\text{sim}}$.)
6. Answer any query to f by querying f (as before), and answer any query to q by querying $p_{\text{sim}} - \tilde{c}_{\text{sim}} \cdot f$.
7. Simulate the interaction of $P_{\text{SUM}}(\mathbb{x}', p_{\text{sim}})$ and $\tilde{V}$.

Note that S runs in polynomial time, and the number of queries it makes to f is exactly the number of queries that $\tilde{V}$ makes to f and q.

To see that $\tilde{V}$'s view is perfectly simulated, we consider a hybrid experiment in which the "hybrid prover" reads all of f (like the honest prover in the real world) but can modify messages after they are sent (like the simulator in the ideal world).

1. Sample $q \in \text{RS}[L, \rho]$ uniformly at random and start simulating $\tilde{V}$.
2. Send q to $\tilde{V}$, along with $\beta := \sum_{a \in H} q(a)$. Let $Q \subseteq L$ be $\tilde{V}$'s queries to q from the beginning of the simulation until the next step.
3. Receive $\tilde{c} \in \mathbb{F}$ from $\tilde{V}$.
4. Sample $p \in \text{RS}[L, \rho]$ uniformly at random such that, for every $\omega \in Q$, $p(\omega) = \tilde{c} \cdot f(\omega) + q(\omega)$ and $\sum_{a \in H} p(a) = \tilde{c} \cdot \mu + \beta$.
5. Replace q with $p - \tilde{c} \cdot f$.
6. Simulate the interaction of $P_{\text{SUM}}(\mathbb{x}', p)$ and $\tilde{V}$.

The distribution of $\tilde{V}$'s view in the real protocol is identical to the distribution of $\tilde{V}$'s view in the above experiment. In particular, all of $\tilde{V}$'s queries to q after its replacement by $p - c \cdot f$ have the correct distribution. Moreover, it is not hard to see that $\tilde{V}$'s view in the above experiment and S's output are identically distributed.

Efficiency. Most of the parameters are seen from the protocol description. We require the prover to send $q \in \text{RS}[L, \rho]$ uniformly at random, which can be done by choosing $\rho|L|$ coefficients uniformly at random and performing one FFT to evaluate that polynomial over L. □

3.4.2 Amortization

Given m instance-witness pairs for univariate sumcheck $\big((\mathbb{F}, L, H, \rho_i, \mu_i), f_i\big)_{i \in [m]}$, we want to test that all of them are in $\mathcal{R}_{\text{SUM}}$. This is achieved with an ℓ-fold increase in complexity, but we want to do this much more efficiently. This will be crucial in our final protocol. We first state formally the relation we will test.

Definition 3.4.9 (ℓ-sumcheck relation)**.** *The relation $\mathcal{R}^{\ell}_{\text{SUM}}$ is the set of all ℓ-tuples*

$$\big((\mathbb{x}_1, \ldots, \mathbb{x}_\ell), (f_1, \ldots, f_\ell)\big)$$

such that, for every $i \in \{1, \ldots, \ell\}$, $\mathbb{x}_i = (\mathbb{F}, L, H, \rho_i, \mu_i)$, and $(\mathbb{x}_i, f_i) \in \mathcal{R}_{\text{SUM}}$.

The idea is to have the verifier choose $z_1, \ldots, z_m \in \mathbb{F}$ uniformly at random and send them to the prover, and then to test that $\sum_{a \in H} \sum_{i=1}^{m} z_i f_i(a) = \sum_{i=1}^{m} z_i \mu_i$. Completeness is easy to see, and soundness follows from properties of random linear combinations. The verifier runtime is increased only by an additive m term, which corresponds to sending $z_1, \ldots, z_m$ and querying each f_i in one position. Crucially, the proof length is unchanged, and the prover still only performs three FFTs. We obtain the following lemma.

Lemma 3.4.10. *There is an RS-encoded IOPP for the univariate m-sumcheck relation (Definition 3.4.9) with the following parameters:*

$$\left|\begin{array}{lllll} \textit{alphabet} & \Sigma & = & \mathbb{F} \\ \textit{number of rounds} & \mathsf{k} & = & 1 \\ \textit{proof length} & \mathsf{p} & = & |L| \\ \textit{randomness} & \mathsf{r} & = & m \log |\mathbb{F}| \\ \textit{soundness error} & \varepsilon & = & 1/|\mathbb{F}| \\ \textit{prover time} & \mathsf{t_P} & = & O(|L| \log |H| + m \cdot |L|) + 2 \cdot \mathrm{FFT}(\mathbb{F}, |L|) \\ \textit{verifier time} & \mathsf{t_V} & = & O(\log^2 |H| + m) \\ \textit{maximum rate} & (\rho_{\mathsf{c}}, \rho_{\mathsf{e}}) & = & (\rho, \rho) \end{array}\right| ,$$

for any instance $\vec{\mathbb{x}} = (\mathbb{x}_1, \ldots, \mathbb{x}_m) = \big((\mathbb{F}, L, H, \rho_i, \mu_i) \big)_{i=1}^{m}$, where $\rho := \max_i \rho_i$.

Protocol 3.4.11. Let $\rho := \max_i \rho_i$, and let $f_1, \ldots, f_m \in \mathrm{RS}[L, \rho]$ be the witness oracles. Let $(P_{\mathrm{SUM}}, V_{\mathrm{SUM}})$ be the standard RS-encoded IOP for univariate sumcheck (Protocol 3.4.3). The RS-encoded IOP protocol for univariate m-sumcheck proceeds as follows.

1. V chooses $z_1, \ldots, z_m \in \mathbb{F}$ uniformly at random, and sends them to P.
2. P and V invoke $\big(P_{\mathrm{SUM}}(\mathbb{x}^*, \mathbb{w}^*), V_{\mathrm{SUM}}^{\mathbb{w}^*}(\mathbb{x}^*)\big)$, where $\mathbb{x}^* := (\mathbb{F}, L, H, \rho, \sum_{i=1}^{m} z_i \mu_i)$, $\mathbb{w}^* := \sum_{i=1}^{m} z_i f_i$.

Proof.

Completeness. Suppose that, for all $i \in [m]$, $(\mathbb{x}_i, f_i) \in \mathcal{R}_{\mathrm{SUM}}$. Then for any choice of $z_1, \ldots, z_m \in \mathbb{F}$, $\sum_{a \in H} \sum_{i=1}^{m} z_i f_i(a) = \sum_{i=1}^{m} z_i \mu_i$, so $(\mathbb{x}^*, \mathbb{w}^*) \in \mathcal{R}_{\mathrm{SUM}}$.

Soundness. Suppose that, for some $i \in [m]$, $(\mathbb{x}_i, f_i) \notin \mathcal{R}_{\mathrm{SUM}}$. Then since $z_1, \ldots, z_m \in \mathbb{F}$ are uniformly random, $\sum_{a \in H} \sum_{i=1}^{m} z_i f_i(a) = \sum_{i=1}^{m} z_i \mu_i$ (i.e., $(\mathbb{x}^*, \mathbb{w}^*) \in \mathcal{R}_{\mathrm{SUM}}$) with probability at most $1/|\mathbb{F}|$.

Efficiency. The efficiency of the system corresponds to a single invocation of univariate sumcheck. The prover, in addition to the cost of running P_{SUM}, pays $O(m \cdot |L|)$ to construct $\mathbb{w}^*$. The verifier pays only an additive $O(m)$ to pick $z_1, \ldots, z_m$ and construct $\mathbb{x}^*$. □

3.5 Univariate lincheck

We describe *univariate lincheck*, an RS-encoded IOPP for verifying linear relations on Reed–Solomon codewords. Given $H_1, H_2 \subseteq \mathbb{F}$, $f_1, f_2 \in \mathrm{RS}[L, \rho]$, and a coefficient matrix $M \in \mathbb{F}^{H_1 \times H_2}$, we want to check that $\hat{f}_1|_{H_1} = M \cdot \hat{f}_2|_{H_2}$, where $\cdot$ is standard matrix multiplication over $\mathbb{F}$. The next definition captures this.

Definition 3.5.1 (lincheck relation)**.** *The relation $\mathcal{R}_{\text{LIN}}$ is the set of all pairs*

$$\left((\mathbb{F}, L, H_1, H_2, \rho, M), (f_1, f_2) \right)$$

where $\mathbb{F}$ is a finite field, L, H_1, H_2 are affine subspaces of $\mathbb{F}$, $\rho \in (0,1)$, $f_1, f_2 \in \text{RS}[L, \rho]$, $M \in \mathbb{F}^{H_1 \times H_2}$, and $\forall\, a \in H_1 \;\; \hat{f}_1(a) = \sum_{b \in H_2} M_{a,b} \cdot \hat{f}_2(b)$.

To build intuition, consider that, given vectors $x \in \mathbb{F}^m, y \in \mathbb{F}^n$ and a matrix $M \in \mathbb{F}^{m \times n}$, a simple probabilistic test for the claim "$x = My$" is to check that $\langle r, x - My \rangle = 0$ for a random $r \in \mathbb{F}^m$. Indeed, if $x \neq My$ then $\Pr_r[\langle r, x - My \rangle = 0] = 1/|\mathbb{F}|$. However, this approach would require the verifier to sample m random field elements, and send these to the prover. A straightforward modification (used also, e.g., in [16, §5.2]) requires only a *single* random field element and incurs only a modest increase in soundness error. Namely, letting $h(X) := \langle \vec{X}, x - My \rangle$ where $\vec{X} := (1, X, \ldots, X^{m-1})$, if $x \neq My$ then $h(X)$ is a non-zero polynomial of degree less than m over $\mathbb{F}$, and thus $\Pr_{\alpha \in \mathbb{F}}[h(\alpha) = 0] \leq m/|\mathbb{F}|$. The verifier now merely has to sample and send $\alpha \in \mathbb{F}$, and the prover must then prove the claim "$h(\alpha) = 0$" to the verifier. This latter claim is in fact a claim about *sums*: one can rewrite $h(X)$ as $\langle \vec{X}, x \rangle - \langle M^\top \vec{X}, y \rangle$ and, expanding the inner products, we obtain the two-sum expression $h(\alpha) = \sum_{i=1}^{m} \alpha^{i-1} x_i - \sum_{j=1}^{n} (\sum_{i=1}^{m} M_{i,j} \alpha^{i-1}) y_j$.

We now return to the RS-encoded version of the problem (defined above), and explain how the prover can handle the claim "$h(\alpha) = 0$" via the univariate sumcheck protocol.

We can think of $\hat{f}_1$ and $\hat{f}_2$ as the low-degree extensions of some $x \in \mathbb{F}^{H_1}$ and $y \in \mathbb{F}^{H_2}$ with $m := |H_1|$ and $n := |H_2|$. The verifier samples and sends $\alpha \in \mathbb{F}$ to the prover; the prover and verifier each compute the low-degree extension $\hat{p}_\alpha^{(1)}$ of $\vec{\alpha} := (1, \alpha, \ldots, \alpha^{m-1})$, and the low-degree extension $\hat{p}_\alpha^{(2)}$ of $M^\top \vec{\alpha}$. We can then write $h(\alpha) = \sum_{a \in H_1} \hat{p}_\alpha^{(1)}(a) \hat{f}_1(a) - \sum_{b \in H_2} \hat{p}_\alpha^{(2)}(b) \hat{f}_2(b)$. In sum, we reduced the claim "$h(\alpha) = 0$" to a sumcheck instance of the polynomial $\hat{p}_\alpha^{(1)}(\cdot) \hat{f}_1(\cdot)$ over H_1 and one of the polynomial $\hat{p}_\alpha^{(2)}(\cdot) \hat{f}_2(\cdot)$ over H_2.

While $h(\alpha)$ equals zero in the honest case, the value of each summation may reveal information. Therefore, to ensure zero knowledge, we combine these two summations into a single summation over the affine space $H_1 \diamond H_2$, defined to be the smallest affine space that contains both H_1 and H_2 (and note that if H_1, H_2 are linear subspaces then $H_1 \diamond H_2 = H_1 + H_2$). Since the precise choice of H_1, H_2 is not important, for efficiency we will typically choose $H_1 \subseteq H_2$ or $H_2 \subseteq H_1$ in order to minimize $|H_1 \diamond H_2|$.

Theorem 3.5.2. *Protocol 3.5.3 below is an RS-encoded IOPP for $\mathcal{R}_{\text{LIN}}$ (Definition 3.5.1) with parameters:*

alphabet	Σ	$=$	$\mathbb{F}$
number of rounds	k	$=$	1
proof length	p	$=$	$\lvert L \rvert$
randomness	r	$=$	$\log \lvert \mathbb{F} \rvert$
soundness error	ε	$=$	$\lvert H_1 \rvert / \lvert \mathbb{F} \rvert$
prover time	$\mathsf{t_P}$	$=$	$O(\lVert M \rVert + \lvert L \rvert \log s) + 4 \cdot \text{FFT}(\mathbb{F}, \lvert L \rvert) + \mathsf{t}(P_{\text{SUM}}; \mathbb{F}, \lvert L \rvert, s)$
verifier time	$\mathsf{t_V}$	$=$	$O(\lVert M \rVert + s) + \mathsf{t}(V_{\text{SUM}}; \mathbb{F}, \lvert L \rvert, s)$
maximum rate	$(\rho_{\mathsf{c}}, \rho_{\mathsf{e}})$	$=$	$(\rho, \rho + s/\lvert L \rvert)$

,

where $s := |H_1 \diamond H_2|$ *and* $\|M\|$ *is the number of nonzero entries of* M*.*

Protocol 3.5.3. Let $f_1, f_2 \in \mathrm{RS}[L, \rho]$ be the witness oracles, and let $\hat{f}_1, \hat{f}_1$ be the unique polynomials of degree at most $\rho|L|$ that agree with f_1, f_2. For every $\alpha \in \mathbb{F}$, we define $q_\alpha \in \mathrm{RS}[L, \rho + |H_1 \diamond H_2|/|L|]$ to be the codeword obtained by evaluating on L the following polynomial:

$$\hat{q}_\alpha(X) := \hat{f}_1(X)\hat{p}^{(1)}_\alpha(X) - \hat{f}_2(X)\hat{p}^{(2)}_\alpha(X) \ ,$$

where

- $\hat{p}^{(1)}_\alpha$ is the unique polynomial of degree less than $|H_1 \diamond H_2|$ s.t. $\hat{p}^{(1)}_\alpha(a) = \alpha^{\gamma(a)}$ for all $a \in H_1$, and $\hat{p}^{(1)}_\alpha(b) = 0$ for all $b \in H_1 \diamond H_2 \setminus H_1$;
- $\hat{p}^{(2)}_\alpha$ is the unique polynomial of degree less than $|H_1 \diamond H_2|$ s.t. $\hat{p}^{(2)}_\alpha(b) = \sum_{a \in H_1} M_{a,b} \cdot \alpha^{\gamma(a)}$ for all $b \in H_2$, and $\hat{p}^{(2)}_\alpha(a) = 0$ for all $a \in H_1 \diamond H_2 \setminus H_2$.

Denote by $(P_{\mathrm{SUM}}, V_{\mathrm{SUM}})$ the RS-encoded IOPP for univariate sumcheck (Protocol 3.4.3). The RS-encoded IOPP (P, V) for $\mathcal{R}_{\mathrm{LIN}}$ works as follows.

1. P and V agree in advance on an ordering $\gamma\colon H_1 \to \{0, \ldots, |H_1| - 1\}$ of H_1.
2. V draws a uniformly random $\alpha \in \mathbb{F}$ and sends it to P. This, along with the witness oracles f_1 and f_2, defines the polynomial q_α. Note that V can use its oracles f_1, f_2 to simulate access to the oracle q_α.
3. P and V run $(P_{\mathrm{SUM}}(\mathbb{x}', q_\alpha), V^{q_\alpha}_{\mathrm{SUM}}(\mathbb{x}'))$ where $\mathbb{x}' := (\mathbb{F}, L, H_1 \diamond H_2, \rho + |H_1 \diamond H_2|/|L|, \mu = 0)$.
4. V accepts if and only if V_{SUM} accepts.

Proof. Completeness and soundness rely on the fact that, by rearranging terms, for every $\alpha \in \mathbb{F}$ it holds that:

$$\begin{aligned} h(\alpha) := \sum_{b \in H_1 \diamond H_2} \hat{q}_\alpha(b) &= \sum_{a \in H_1} \hat{f}_1(a)\alpha^{\gamma(a)} - \sum_{b \in H_2} \sum_{a \in H_1} M_{a,b}\hat{f}_2(b)\alpha^{\gamma(a)} \\ &= \sum_{a \in H_1} \left(\hat{f}_1(a) - \sum_{b \in H_2} M_{a,b} \cdot \hat{f}_2(b) \right) \cdot \alpha^{\gamma(a)} \ . \end{aligned}$$

Completeness. Suppose that, for all $a \in H_1$, $\hat{f}_1(a) = \sum_{b \in H_2} M_{a,b} \cdot \hat{f}_2(b)$. For every $\alpha \in \mathbb{F}$, $h(\alpha) = 0$ and thus $\sum_{b \in H_2} \hat{q}_\alpha(b) = 0$. Completeness of the univariate sumcheck implies that V_{SUM} always accepts.

Soundness. Suppose that there exists $a \in H_1$ such that $\hat{f}_1(a) \neq \sum_{b \in H_2} M_{a,b} \cdot \hat{f}_2(b)$. This implies that h is a nonzero polynomial of degree less than $|H_1|$, and so $\Pr_{\alpha \in \mathbb{F}}[h(\alpha) = 0] < |H_1|/|\mathbb{F}|$. If $h(\alpha) \neq 0$, then $\sum_{b \in H_2} \hat{q}_\alpha(b) \neq 0$ and in this case V_{SUM} rejects.

Efficiency. Both parties run the univariate sumcheck as a subroutine. In addition, the prover needs to compute $q_\alpha = \hat{q}_\alpha|_L$ (the evaluation of $\hat{q}_\alpha$ over L), for example as follows: (i) evaluate $\hat{p}^{(1)}_\alpha$ over L in time $O(|H_1 \diamond H_2|) + 2 \cdot \mathrm{FFT}(\mathbb{F}, |L|)$; (ii) evaluate $\hat{p}^{(2)}_\alpha$ over L in time $O(\|M\| + |H_1 \diamond H_2|) + 2 \cdot \mathrm{FFT}(\mathbb{F}, |L|)$; (iii) compute q_α from these components in time $O(|L|)$. The verifier needs to construct a circuit that simulates oracle access to q_α, which can be done in time $O(\|M\| + |H_1 \diamond H_2|)$, and then run V_{SUM}. □

3.6 An RS-encoded IOP for rank-one constraint satisfaction

We describe an RS-encoded IOP for rank-one constraint satisfaction (R1CS). An R1CS instance consists of matrices $A, B, C \in \mathbb{F}^{m \times (n+1)}$ and explicit input $v \in \mathbb{F}^k$, and it is satisfiable if there exists $w \in \mathbb{F}^{n-k}$ such that $Az \circ Bz = Cz$ where $z = (1, v, w) \in \mathbb{F}^{n+1}$ and $\circ$ denotes entry-wise (Hadamard) product.

Definition 3.6.1 (R1CS relation)**.** *The relation $\mathcal{R}_{\mathrm{R1CS}}$ is the set of all pairs $\big((I, v), w\big)$ where $\mathbb{F}$ is a finite field, $k, n, m \in \mathbb{N}$ denote the number of inputs, variables and constraints respectively ($k \leq n$), A, B, C are $m \times (1 + n)$ matrices over $\mathbb{F}$, $v \in \mathbb{F}^k$, and $w \in \mathbb{F}^{n-k}$, such that for all $i \in [m]$ $\big(\sum_{j=0}^{n} A_{i,j} z_j\big) \cdot \big(\sum_{j=0}^{n} B_{i,j} z_j\big) = \big(\sum_{j=0}^{n} C_{i,j} z_j\big)$, where $z := (1, v, w) \in \mathbb{F}^{n+1}$.*

We describe how to obtain an RS-encoded IOP for R1CS by using RS-encoded IOPPs for lincheck (which we obtained in Section 3.5) and rowcheck (see Section 2.3.4).

Let H_1, H_2 be subspaces of $\mathbb{F}$ such that $|H_1| = m$ and $|H_2| = n + 1$, and view A, B, C as matrices in $\mathbb{F}^{H_1 \times H_2}$. The prover first sends four oracles: f_z that (purportedly) is the low-degree extension of $z\colon H_2 \to \mathbb{F}$; and f_{Az}, f_{Bz}, f_{Cz} that (purportedly) are the low-degree extensions of $Az, Bz, Cz\colon H_1 \to \mathbb{F}$. The verifier uses the lincheck protocol to test that, indeed, f_{Az} is a low-degree extension of Az, and likewise for f_{Bz}, f_{Cz}. Then the verifier uses the rowcheck protocol to check that $f_{Az}(a) \cdot f_{Bz}(a) = f_{Cz}(a)$ for all $a \in H_1$.

The above protocol almost works, with the one problem being that the prover could cheat by sending f_z that is inconsistent with the explicit input v. We remedy this by (roughly) having the prover send the low-degree extension f_w of w instead of f_z. The verifier only needs to query one point of f_z, which it can do by making one query to f_w and evaluating the low-degree extension of v at one point.

The above protocol uses three linchecks and one rowcheck. Each lincheck is a probabilistic reduction to sumcheck; this means running the sumcheck protocol three times (in parallel). The sumcheck protocol is relatively expensive, so we use the optimization of bundling these sumcheck instances (see Section 3.4.2). We also save computation by choosing the same challenge α for each of the linchecks.

Below we provide details about the foregoing intuition. After that we provide additional subsections that explain how to modify the "basic" protocol to achieve additional goals: in Section 3.6.1 we describe how to achieve zero knowledge; in Section 3.6.2 we describe how to amortize the cost of verifying the satisfaction of multiple R1CS instances (sharing the same matrices) at the same time.

Theorem 3.6.2. *Protocol 3.6.3 below is an RS-encoded IOP of knowledge for $\mathcal{R}_{\mathrm{R1CS}}$ (Defini-*

tion 3.6.1) with parameters:

$$\left|\begin{array}{lllll} \textit{alphabet} & \Sigma & = & \mathbb{F} \\ \textit{number of rounds} & \mathsf{k} & = & 2 \\ \textit{proof length} & \mathsf{p} & = & 5|L| \\ \textit{randomness} & \mathsf{r} & = & O(\log|\mathbb{F}|) \\ \textit{soundness error} & \varepsilon & = & \frac{m+1}{|\mathbb{F}|} \\ \textit{prover time} & \mathsf{t_P} & = & O(|L|\cdot\log(n+m)+\|A\|+\|B\|+\|C\|) \\ & & & \quad +17\cdot\mathrm{FFT}(\mathbb{F},|L|) \\ \textit{verifier time} & \mathsf{t_V} & = & O(\|A\|+\|B\|+\|C\|+n+m) \\ \textit{maximum rate} & (\rho_{\mathsf{c}},\rho_{\mathsf{e}}) & = & (\frac{\max(m,n+1)}{|L|},\frac{2\max(m,n+1)}{|L|}) \end{array}\right| ,$$

for any instance $\mathbb{x}=(I,v)$ *and any affine subspace* L *of* $\mathbb{F}$.

Protocol 3.6.3. The prover P and verifier V both receive as input an R1CS instance (I,v), and the prover P also receives as input a corresponding R1CS witness w; as above, $z:=(1,v,w)\in\mathbb{F}^{n+1}$.

Below, $(P_{\text{LIN}},V_{\text{LIN}})$ denotes the RS-encoded IOPP for univariate lincheck (Protocol 3.5.3).

Let H_1,H_2 be two affine subspaces of $\mathbb{F}$ with $|H_1|=m$ and $|H_2|=n+1$ such that $H_1\subseteq H_2$ or $H_2\subseteq H_1$; this implies that $H_1\diamond H_2=H_1\cup H_2$. (We assume without loss of generality that m, $n+1$, and $k+1$ are powers of $\mathrm{char}(\mathbb{F})$.) Let $\gamma\colon H_1\cup H_2\to\{0,\ldots,|H_1\cup H_2|-1\}$ be an ordering on $H_1\cup H_2$ such that $\gamma(H_i)=\{0,\ldots,|H_i|-1\}$ for $i\in\{1,2\}$. We view A,B,C as matrices in $\mathbb{F}^{H_1\times H_2}$ via this ordering.

1. **Compute LDE of the input.** Letting $H_2^{\le k}:=\{b\in H_2:0\le\gamma(b)\le k\}$, P and V construct $\hat{f}_{(1,v)}(X)$, the unique polynomial of degree less than $|H_2^{\le k}|=k+1$ such that, for all $b\in H_2^{\le k}$,

$$\hat{f}_{(1,v)}(b)=\begin{cases}1 & \text{if } \gamma(b)=0,\\ v_i & \text{if } \gamma(b)=i \text{ and } i\in\{1,\ldots,k\}.\end{cases}$$

2. **Witness and auxiliary oracles.** P sends to V the oracle codewords $f_w\in\mathrm{RS}[L,\frac{n-k}{|L|}]$ and $f_{Az},f_{Bz},f_{Cz}\in\mathrm{RS}[L,\frac{m}{|L|}]$ defined as follows.
   - $f_w:=\hat{f}_w|_L$ where $\hat{f}_w$ is the unique polynomial of degree less than $n-k$ such that

$$\forall\, b\in H_2 \text{ with } k<\gamma(b)\le n,\quad \hat{f}_w(b)=\frac{w_{\gamma(b)-k}-\hat{f}_{(1,v)}(b)}{\mathbb{Z}_{H_2^{\le k}}(b)}\ .$$

   - $f_{Az}:=\hat{f}_{Az}|_L$ where $\hat{f}_{Az}$ is the unique polynomial of degree less than m such that, for all $a\in H_1$, $\hat{f}_{Az}(a)=\sum_{b\in H_2}A_{a,b}\cdot z_{\gamma(b)}=(Az)_a$. The other codewords, f_{Bz} and f_{Cz}, are defined similarly.

   These implicitly define the "virtual oracle" $f_z:=\hat{f}_z|_L$ where $\hat{f}_z(X):=\hat{f}_w(X)\cdot\mathbb{Z}_{H_2^{\le k}}(X)+\hat{f}_{(1,v)}(X)$. Note that $\hat{f}_z(b)=z_{\gamma(b)}$ for all $b\in H_2$, and $f_z\in\mathrm{RS}[L,\frac{n+1}{|L|}]$.
3. **Run subprotocols.** P and V run the following in parallel:
   (a) $(P_{\text{LIN}}(\mathbb{x}^A_{\text{LIN}},(f_{Az},f_z)),V^{f_{Az},f_z}_{\text{LIN}}(\mathbb{x}^A_{\text{LIN}}))$ with $\mathbb{x}^A_{\text{LIN}}:=(\mathbb{F},L,H_1,H_2,|H_1\cup H_2|/|L|,A)$.

(b) $(P_{\text{LIN}}(\mathfrak{x}^B_{\text{LIN}}, (f_{Bz}, f_z)), V^{f_{Bz}, f_z}_{\text{LIN}}(\mathfrak{x}^B_{\text{LIN}}))$ with $\mathfrak{x}^B_{\text{LIN}} := (\mathbb{F}, L, H_1, H_2, |H_1 \cup H_2|/|L|, B)$.
(c) $(P_{\text{LIN}}(\mathfrak{x}^C_{\text{LIN}}, (f_{Cz}, f_z)), V^{f_{Cz}, f_z}_{\text{LIN}}(\mathfrak{x}^C_{\text{LIN}}))$ with $\mathfrak{x}^C_{\text{LIN}} := (\mathbb{F}, L, H_1, H_2, |H_1 \cup H_2|/|L|, C)$.
(d) Check $p \in \text{RS}[L, (|H_1| - 1)/|L|]$ for $\hat{p}(X) := (\hat{f}_{Az}(X) \cdot \hat{f}_{Bz}(X) - \hat{f}_{Cz}(X))/\mathbb{Z}_{H_1}(X)$. In the formalism of RS-encoded IOPs (see Section 2.3.3), this corresponds to the rational constraint $(\mathcal{C}, \sigma) := ((N, D), (|H_1| - 1)/|L|)$ where $N(X, Z_1, Z_2, Z_3) := Z_1 \cdot Z_2 - Z_3$ and $D(X) := \mathbb{Z}_{H_1}(X)$. This realizes the required rowcheck (see Section 2.3.4).

4. V accepts if and only if all of the above subverifiers accept.

Note that the maximum rate across the protocol is $\rho_{\mathsf{e}} = 2|H_1 \cup H_2|/|L|$.

Proof.

Completeness. Suppose that $w \in \mathbb{F}^{n-k}$ is a valid witness for the instance (I, v), and define $z := (1, v, w) \in \mathbb{F}^{n+1}$. By construction, $\hat{f}_z$ is a low-degree extension of z over H_2 (i.e., $\hat{f}_z(b) = z_{\gamma(b)}$ for all $b \in H_2$). Therefore, for all $a \in H_1$ it holds that $\hat{f}_A(a) = \sum_{b \in H_2} A_{a,b} z_{\gamma(b)} = \sum_{b \in H_2} A_{a,b} \hat{f}_z(b)$, and so Step 3a (lincheck on (f_{Az}, f_z)) always accepts. By the same argument, Steps 3b and 3c always accept. Finally, the fact that w is a valid witness implies that, for all $a \in H_1$, it holds that $\hat{f}_A(a) \cdot \hat{f}_B(a) - \hat{f}_C(a) = (\sum_{b \in H_2} A_{a,b} z_{\gamma(b)}) \cdot (\sum_{b \in H_2} B_{a,b} z_{\gamma(b)}) - (\sum_{b \in H_2} C_{a,b} z_{\gamma(b)}) = 0$, which means that Step 3d always accepts.

Soundness. Suppose that the instance (I, v) is not satisfiable, i.e., for all $w \in \mathbb{F}^{n-k}$, letting $z := (1, v, w)$, there exists $i \in [m]$ such that $(\sum_{j=0}^{n} A_{i,j} z_j) \cdot (\sum_{j=0}^{n} B_{i,j} z_j) \neq (\sum_{j=0}^{n} C_{i,j} z_j)$. Let the oracles sent by a malicious prover be f'_w, f'_A, f'_B, f'_C, and let $f'_z := \hat{f}'_z|_H$ where $\hat{f}'_z(X) := \hat{f}'_w(X) \cdot \mathbb{Z}_{H_2^{\leq k}}(X) + \hat{f}_{(1,v)}(X)$. We distinguish between multiple cases.

(i) There exists $a \in H_1$ for which $\hat{f}'_A(a) \neq \sum_{b \in H_2} A_{a,b} \hat{f}'_z(b)$. Then, by soundness of the lincheck protocol, Step 3a accepts with probability $\frac{|H_1|}{|\mathbb{F}|}$.
(ii) If analogous statements hold for f'_B or f'_C, then analogous conclusions hold for Step 3b or Step 3c.
(iii) For all $a \in H_1$ it holds that $\hat{f}'_A(a) = \sum_{b \in H_2} A_{a,b} \hat{f}'_z(b)$, and likewise for $\hat{f}'_B, \hat{f}'_C$. Then by assumption there exists $a \in H_1$ such that $f'_A(a) \cdot f'_B(a) - f'_C(a) \neq 0$. We conclude that $\hat{f}'_{Az}(X) \cdot \hat{f}'_{Bz}(X) - \hat{f}'_{Cz}(X)$ is not divisible by $\mathbb{Z}_{H_1}(X)$, which implies that the ratio of these two polynomials does not yield a codeword $p \in \text{RS}[L, (|H_1| - 1)/|L|]$, so the verifier rejects (with probability 1).

The soundness error is given by maximizing over the above cases.

Next, we discuss two optimizations that come from viewing the lincheck protocols as probabilistic interactive reductions to sumcheck: given an instance-witness pair

$$\big((\mathbb{F}, L, H_1, H_2, \rho, M), (f_1, f_2) \big)$$

the lincheck protocol outputs an instance-witness pair

$$\big((\mathbb{F}, L, H_1 \cup H_2, \rho + |H_1 \cup H_2|/|L|, 0), q_\alpha \big)$$

for sumcheck.

Optimization: one sumcheck suffices. All lincheck summations are taken over the same space, so we can save costs by running a single execution of the *amortized* sumcheck protocol (see Lemma 3.4.10 in Section 3.4.2).

Optimization: re-use α. Each lincheck protocol instructs the verifier to send a random $\alpha \in \mathbb{F}$ to the prover. The verifier thus chooses $\alpha_1, \alpha_2, \alpha_3 \in \mathbb{F}$ uniformly and independently, and sends $(\alpha_1, \alpha_2, \alpha_3)$ to the prover. But this means that the verifier must compute $\hat{p}_{\alpha_i}$ for each i. We observe that choosing $\alpha_1 = \alpha_2 = \alpha_3 \in \mathbb{F}$ uniformly at random does not affect soundness, which means that the verifier only has to compute $\hat{p}_\alpha$ for one α. This will become more important later in Section 3.6.2 when we consider amortizing multiple instances.

Efficiency. The prover computes Az, Bz, Cz and their low-degree extensions, along with the low-degree extensions of w and z, in time $O(\|A\| + \|B\| + \|C\| + n + m) + 5 \cdot \mathrm{FFT}(\mathbb{F}, |L|)$. The verifier evaluates f_z at a single point in L, which costs $O(n + m)$. Summing over the costs of the subprotocols and amortizing the sumcheck cost across the four instances yields the stated expressions.

Proof of knowledge. Suppose that $\tilde{P}$ causes the verifier's rational constraints to be satisfied with probability at least μ. Then by the soundness analysis with probability at least $\mu - \frac{m+1}{\mathbb{F}}$ it holds that $\hat{f}_A(a) \cdot \hat{f}_B(a) - \hat{f}_C(a) = (\sum_{b \in H_2} A_{a,b} \hat{f}_z(b)) \cdot (\sum_{b \in H_2} B_{a,b} \hat{f}_z(b)) - (\sum_{b \in H_2} C_{a,b} \hat{f}_z(b)) = 0$. Hence it holds that f_w encodes a valid witness w for $\mathbb{x}$. The extractor queries f_w at sufficiently many points to decode w. □

3.6.1 Zero knowledge

We describe how to modify Protocol 3.6.3 to achieve zero knowledge against bounded-query malicious verifiers; the modification is an adaptation of algebraic techniques from [26, 24]. Essentially, instead of providing the *unique* low-degree extensions of w, Az, Bz, Cz, the prover provides *randomized* low-degree extensions that are over a domain $L \subseteq \mathbb{F}$ chosen such that $(H_1 \cup H_2) \cap L = \emptyset$ (in particular, L will be *affine* so that $0_{\mathbb{F}} \notin L$). This ensures that a bounded number of queries to the witness and auxiliary oracles does not reveal any information about w. Then, both prover and verifier use our zero knowledge sumcheck protocol (see Protocol 3.4.8 in Section 3.4.1) instead of the "plain" sumcheck protocol used above.

Theorem 3.6.4. *For any* $\mathsf{b} \colon \mathbb{N} \to \mathbb{N}$*, Protocol 3.6.5 below is an RS-encoded IOP of knowledge for* $\mathcal{R}_{\mathrm{R1CS}}$ *(Definition 3.6.1) that is zero knowledge against query bound* b *with parameters:*

alphabet	Σ	$=$	$\mathbb{F}$
number of rounds	k	$=$	2
proof length	p	$=$	$6\|L\|$
randomness	r	$=$	$O(\log \|\mathbb{F}\|)$
soundness error	ε	$=$	$\frac{m+1}{\|\mathbb{F}\|}$
prover time	$\mathsf{t_P}$	$=$	$O(\|L\| \cdot \log(n+m) + \|A\| + \|B\| + \|C\|) + 18 \cdot \mathrm{FFT}(\mathbb{F}, \|L\|)$
verifier time	$\mathsf{t_V}$	$=$	$O(\|A\| + \|B\| + \|C\| + n + m)$
maximum rate	$(\rho_{\mathsf{c}}, \rho_{\mathsf{e}})$	$=$	$(\frac{2\max(m,n+1)+2\mathsf{b}}{\|L\|}, \frac{2\max(m,n+1)+2\mathsf{b}}{\|L\|})$

for any instance $\mathbb{x} = (I, v)$.

Protocol 3.6.5 (ZK variant of Protocol 3.6.3)**.** We use the same notation as in Protocol 3.6.3, with the only additional constraint that $(H_1 \cup H_2) \cap L = \varnothing$.

1. **Compute LDE of the input.** Same as Step 1 in Protocol 3.6.3.
2. **Witness and auxiliary oracles.** P sends to V the oracle codewords $f_w \in \mathrm{RS}[L, \frac{n-k+\mathsf{b}}{|L|}]$ and $f_{Az}, f_{Bz}, f_{Cz} \in \mathrm{RS}[L, \frac{m+\mathsf{b}}{|L|}]$ defined as follows.
   - $f_w := \bar{f}_w|_L$ where $\bar{f}_w$ is a *random* polynomial of degree less than $n-k+\mathsf{b}$ such that

$$\forall\, b \in H_2 \text{ with } k < \gamma(b) \leq n, \quad \bar{f}_w(b) = \frac{w_{\gamma(b)-k} - \hat{f}_{(1,v)}(b)}{\mathbb{Z}_{H_2^{\leq k}}(b)} \ .$$

   - $f_{Az} := \bar{f}_{Az}|_L$ where $\bar{f}_{Az}$ is a *random* polynomial of degree less than $m+\mathsf{b}$ such that, for all $a \in H_1$, $\bar{f}_{Az}(a) = \sum_{b \in H_2} A_{a,b} \cdot z_{\gamma(b)} = (Az)_a$. The other codewords, f_{Bz} and f_{Cz}, are defined similarly.

   As before, the above implicitly define the "virtual oracle" $f_z := \hat{f}_z|_L$ where $\hat{f}_z(X) := \bar{f}_w(X) \cdot \mathbb{Z}_{H_2^{\leq k}}(X) + \hat{f}_{(1,v)}(X)$. Again $\hat{f}_z(b) = z_{\gamma(b)}$ for all $b \in H_2$, but now $f_z \in \mathrm{RS}[L, \frac{n+1+\mathsf{b}}{|L|}]$ since $\bar{f}_w$ has higher degree.
3. **Run subprotocols.** The same as Step 3 in Protocol 3.6.3, except that P and V run the (amortized) zero knowledge sumcheck protocol (see Protocol 3.4.8 in Section 3.4.1). Several rates need to be adjusted to take into account the fact that the codewords encoding the satisfying assignment (and its linear transformations) have degree higher by b. This results in the following maximum rate

$$(\rho_{\mathsf{c}}, \rho_{\mathsf{e}}) = \left(\frac{2|H_1 \cup H_2| + 2\mathsf{b}}{|L|}, \frac{2|H_1 \cup H_2| + 2\mathsf{b}}{|L|} \right) \ .$$

4. V accepts if and only if all of the above subverifiers accept.

Proof. Completeness, soundness, and proof of knowledge follow almost directly from the proof of Theorem 3.6.2, so we do not discuss them. Before discussing zero knowledge, we note that the round complexity can be reduced to 2 by running the first round of the zero knowledge sumcheck protocol (Protocol 3.4.8) in parallel with the first round of Protocol 3.6.5. We now argue the zero knowledge guarantee (see Definition 2.3.3): we need to construct a probabilistic simulator S that, given as input a satisfiable R1CS instance (I, v) and straightline access to a b-query malicious verifier $\tilde{V}$, outputs a view that is identically distributed as $\tilde{V}$'s view when interacting with an honest prover.

At a high level, S simulates the oracles $f_w, f_{Az}, f_{Bz}, f_{Cz}$ by answering each query with uniformly random field elements. Given these, it runs the simulator for the amortized zero knowledge sumcheck, answering the subsimulator's queries to the virtual oracle by "querying" the appropriate locations of $f_w, f_{Az}, f_{Bz}, f_{Cz}$. More precisely, on input $\mathtt{x}$, the simulator operates as follows.

1. Prepare a table T for $(f_{Az}, f_{Bz}, f_{Cz}, f_w)$ which is initially empty. Whenever we "query" an oracle f_x at a point $a \in L$, where x is one of Az, Bz, Cz, w, if $(a, b_{Az}, b_{Bz}, b_{Cz}, b_w) \in T$ then output b_x; otherwise, choose $b_{Az}, b_{Bz}, b_{Cz}, b_w \in \mathbb{F}$ uniformly at random, add $(a, b_{Az}, b_{Bz}, b_{Cz}, b_w)$ to T and output b_x.

2. "Send" the oracles $f_{Az}, f_{Bz}, f_{Cz}, f_w$ to $\tilde{V}$. In parallel, "send" the first prover message r in the univariate ZK sumcheck protocol (Protocol 3.4.8), and use the simulator for that protocol to answer queries to r.

3. Run the prover for each subprotocol in Step 3, except that we do not explicitly construct any witness (we think of them as arithmetic circuits with oracle gates), and we do not run the sumcheck protocol.

4. Pass the instances constructed in the previous step to the simulator for the zero knowledge 3-sumcheck protocol. When the subsimulator queries one of the oracles, evaluate the corresponding circuit and use T to look up the necessary values of $f_{Az}, f_{Bz}, f_{Cz}, f_w$.

The subsimulator makes the same number of queries to the "amortized" virtual oracle as the verifier makes to the sumcheck proof oracle; in particular, this is at most b. By inspecting the virtual oracle, we see that the answer to a query $a \in \mathbb{F}$ depends only on $f_{Az}(a), f_{Bz}(a), f_{Cz}(a), f_w(a)$ (and $f_v(a)$, which is known). Hence $|T| \leq \mathsf{b}$. It remains to show that T is consistent with the real view.

We look at f_w; the other cases are essentially identical. We can write $\bar{f}_w$ as

$$\bar{f}_w := \hat{f}_w(X) + \mathbb{Z}_{H_2^{>k}}(X) \cdot R(X) \ ,$$

where $R(X)$ is a uniformly random polynomial of degree less than b and $H_2^{>k} := H_2 \setminus H_2^{\leq k}$. Since $\mathbb{Z}_{H_2^{>k}}$ is nonzero outside of $H_2^{>k}$, any vector $\left(\bar{f}_w(a)\right)_{a \in Q}$ is distributed uniformly in $\mathbb{F}^Q$ for any $Q \subseteq \mathbb{F}$ such that $Q \cap H_2 = \varnothing$ and $|Q| \leq \mathsf{b}$. In particular, this holds for the set of query positions asked by the subsimulator, even if they are chosen adaptively based on the answers to previous queries. □

3.6.2 Amortization

We describe efficiency savings that can be made when one considers multiple R1CS instances *with the same constraints* (but different inputs). More precisely, we seek an RS-encoded IOP for the following relation:

Definition 3.6.6 (ℓ-wise R1CS relation)**.** *The relation $\mathcal{R}^{\ell}_{\mathrm{R1CS}}$ is the set of all pairs*

$$\left((\mathtt{x}_1, \ldots, \mathtt{x}_\ell), (w_1, \ldots, w_\ell)\right)$$

such that, for every $i \in \{1, \ldots, \ell\}$, $\mathtt{x}_i = (\mathbb{F}, k, n, m, A, B, C, v^{(i)})$ and $(\mathtt{x}_i, w_i) \in \mathcal{R}_{\mathrm{R1CS}}$.

We have already obtained an RS-encoded IOP for $\mathcal{R}_{\mathrm{R1CS}}$ (Protocol 3.6.3), so we can obtain an RS-encoded IOP for $\mathcal{R}^{\ell}_{\mathrm{R1CS}}$ by running this IOP in parallel ℓ times. Note, however, that the running time of both the prover and the verifier increases by a multiplicative factor of ℓ.

We modify this strategy to ensure that the verifier's running time increases by only an *additive* factor in ℓ, for a total of $O(\|A\| + \|B\| + \|C\| + n + m + m)$. This is significant because, as ℓ increases, the amortized per-instance cost becomes constant. The modification follows from an observation used in the proof of Theorem 3.6.2: we choose the same random

$\alpha \in \mathbb{F}$ for all lincheck instances that result from the ℓ parallel executions, and then amortize all of the resulting sumcheck instances (see Section 3.4.2). The verifier then only has to evaluate the auxiliary lincheck polynomials *once*.

Corollary 3.6.7. *For every $m \in \mathbb{N}$ there exists an RS-encoded IOP for $\mathcal{R}^m_{\mathrm{R1CS}}$ (Definition 3.6.6) with parameters:*

alphabet	Σ	$=$	$\mathbb{F}$
number of rounds	k	$=$	2
proof length	p	$=$	$(4m+1)\lvert L\rvert$
randomness	r	$=$	$O(m \log \lvert\mathbb{F}\rvert)$
soundness error	ε	$=$	$\frac{m+1}{\lvert\mathbb{F}\rvert}$
prover time	$\mathsf{t_P}$	$=$	$m \cdot O(\lvert L\rvert \cdot \log(n+m) + \lVert A\rVert + \lVert B\rVert + \lVert C\rVert) + 17 \cdot \mathrm{FFT}(\mathbb{F}, \lvert L\rvert)$
verifier time	$\mathsf{t_V}$	$=$	$O(\lVert A\rVert + \lVert B\rVert + \lVert C\rVert + n + m + m)$
maximum rate	$(\rho_{\mathsf{c}}, \rho_{\mathsf{e}})$	$=$	$(\frac{\max(m,n+1)}{\lvert L\rvert}, \frac{2\max(m,n+1)}{\lvert L\rvert})$

3.7 From RS-encoded provers to arbitrary provers

In prior sections we have designed IOP protocols based on the simplifying assumption that a malicious prover is restricted to sending Reed–Solomon codewords of prescribed rates. In this section we describe how to transform any IOP protocol that is sound under this assumption into one that is sound against all provers.

This by itself should not be surprising: the probabilistic checking literature is rich with such transformations, which are enabled by the tools of *low-degree testing* and *self-correction*. However, our goal here is to obtain a transformation that is particularly efficient for the setting of *this* paper, as we now explain.

There is a straightforward approach to using low-degree testing, which we now spell out since it serves as a comparison point. Suppose that we have a low-degree test for $\mathrm{RS}[L, \rho]$ with soundness error $\varepsilon^{\mathrm{LDT}}$ and proximity parameter δ^{LDT}, and we wish to transform a given RS-encoded IOP $(P, V, (\vec{\rho}_i)_{i=1}^{\mathsf{k}})$ into a corresponding IOP that is sound against all provers. Let us assume for simplicity that $\vec{\rho}_1 = \cdots = \vec{\rho}_{\mathsf{k}} = (\rho)$ for some $\rho \in (0, 1]$, that is, each prover message consists of one codeword in $\mathrm{RS}[L, \rho]$.

The naive approach is to individually run the low-degree test on each prover message. If all tests pass with probability greater than $\varepsilon^{\mathrm{LDT}}$, then every message $\tilde{\pi}_i$ is δ^{LDT}-close to some codeword $\pi_i \in \mathrm{RS}[L, \rho]$. If the verifier makes q uniform queries, the probability that any one of these queries does not "see" $(\pi_i)_{i=1}^{\mathsf{k}}$ is at most $\mathsf{q} \cdot \delta^{\mathrm{LDT}}$. Conditioned on the verifier "seeing" $(\pi_i)_{i=1}^{\mathsf{k}}$, the verifier's acceptance probability is exactly the same as in the RS-encoded protocol.

While the foregoing approach "works", it has two inefficiencies. First, it runs one low-degree test for each purported codeword, which is undesirable because low-degree tests are expensive. Second, the soundness error of the RS-encoded IOP typically decreases by increasing q, which creates a trade-off with the soundness error $\mathsf{q} \cdot \delta^{\mathrm{LDT}}$ of the transformation.

We address the first problem by testing a random linear combination of the π_i, following an idea introduced in [130] (in the context of interactive proofs of proximity) and applied in [9] (in

the context of interactive PCPs of proximity). The verifier samples $a_1, \ldots, a_{\mathsf{k}} \in \mathbb{F}$ uniformly and independently at random, and sends these to the prover; the prover and verifier then engage in a low-degree test for the "virtual oracle" $\tilde{\pi} := \sum_{i=1}^{\mathsf{k}} a_i \tilde{\pi}_i$. If $\tilde{\pi}_i \in \mathrm{RS}[L, \rho]$ for all i, then $\tilde{\pi} \in \mathrm{RS}[L, \rho]$. If instead $\tilde{\pi}_i$ is δ-far from $\mathrm{RS}[L, \rho]$ for some i (and δ small enough), then one can show that $\tilde{\pi}$ is also δ-far with high probability. Thus, a single low-degree test is run, regardless of the number of oracles k.

We address the second problem by using an observation due to [23] about testing *rational constraints* (see Section 2.3.3). In the encoded protocols in this work (and in [23]), soundness entails testing both that the prover's messages are low-degree and that they satisfy some existentially-quantified polynomial equations; for example, "message f is low-degree and there is a low-degree g such that $f \equiv g \cdot \mathbb{Z}_H$". The standard way to test this property is for the prover to send g; the verifier can then check the relation by querying at a uniformly random point in the domain, but this creates the aforementioned trade-off. However, [23] observe that the verifier can *simulate* queries to g itself, given query access to f, since $g(\alpha) = f(\alpha)/\mathbb{Z}_H(\alpha)$ (when $\mathbb{Z}_H(\alpha) \neq 0$). Thus the prover does not have to send g, but only has to show that g is low-degree. In all of our protocols, this observation results in RS-encoded IOPs with $\mathsf{q} = 0$, and we will assume that this is the case in the transformation described in this section.

In this exposition we have made the simplifying assumption that the desired rate for each codeword in each proof is the same. In our protocols (in particular, the sumcheck protocol) this will not be the case, and so we must also handle differing rates. In some settings it suffices to test for proximity to $\mathrm{RS}[L, \max_i \rho_i]^{\mathsf{k}}$, but not in our setting. This is because the soundness of univariate sumcheck relies on g being close to $\mathrm{RS}[L, (|H| - 1)/|L|]$; soundness breaks if g is merely close to (say) $\mathrm{RS}[L, |H|/|L|]$ (or RS codes with bigger rates). Following [41], we instead multiply each $\tilde{\pi}_i$ by an appropriately-chosen monomial and then take a random linear combination. We show that if $\tilde{\Pi}$ is δ-far from $\mathrm{RS}[L, (\rho_1, \ldots, \rho_{\mathsf{k}})]$ then with high probability $\tilde{\pi}$ is far from $\mathrm{RS}[L, \max_i \rho_i]$, which suffices for soundness. We obtain the following theorem.

Theorem 3.7.1. *Suppose that we are given:*
- *an RS-encoded IOP* $(P_{\mathcal{R}}, V_{\mathcal{R}}, (\vec{\rho}_i)_{i=1}^{\mathsf{k}^{\mathcal{R}}})$*, with maximum rate* $(\rho_{\mathsf{c}}, \rho_{\mathsf{e}})$*, for a relation* $\mathcal{R}$*;*
- *an IOPP* $(P_{\mathrm{LDT}}, V_{\mathrm{LDT}})$ *for the RS code* $\mathrm{RS}[L, \rho_{\mathsf{c}}]$.

Then we can combine these two ingredients to obtain an IOP (P, V) *for* $\mathcal{R}$ *with the following parameters:*

$$\left|\begin{array}{lllll} \textit{alphabet} & \Sigma & = & \Sigma^{\mathcal{R}} \\ \textit{number of rounds} & \mathsf{k} & = & \mathsf{k}^{\mathcal{R}} + \mathsf{k}^{\mathrm{LDT}} \\ \textit{proof length} & \mathsf{p} & = & \mathsf{p}^{\mathcal{R}} + \mathsf{p}^{\mathrm{LDT}} \\ \textit{query complexity} & \mathsf{q}_\pi & = & \mathsf{q}_\pi^{\mathrm{LDT}} + \mathsf{q}_{\mathsf{w}}^{\mathrm{LDT}} \cdot \sum_{i=1}^{\mathsf{k}} \ell_i^{\mathcal{R}} \\ \textit{randomness} & (\mathsf{r}_{\mathsf{i}}, \mathsf{r}_{\mathsf{q}}) & = & \left(\mathsf{r}_{\mathsf{i}}^{\mathcal{R}} + \mathsf{r}_{\mathsf{i}}^{\mathrm{LDT}} + (\sum_{i=1}^{\mathsf{k}} \ell_i^{\mathcal{R}} + c) \log |\mathbb{F}|, \mathsf{r}_{\mathsf{q}}^{\mathrm{LDT}}\right) \\ \textit{soundness error} & (\varepsilon_{\mathsf{i}}, \varepsilon_{\mathsf{q}}) & = & \left(\varepsilon^{\mathcal{R}} + \frac{|L|}{|\mathbb{F}|} + \varepsilon_{\mathsf{i}}^{\mathrm{LDT}}, \varepsilon_{\mathsf{q}}^{\mathrm{LDT}}\right) \\ \textit{prover time} & \mathsf{t}_{\mathsf{P}} & = & O(\mathsf{t}_{\mathsf{P}}^{\mathcal{R}} + \mathsf{t}_{\mathsf{P}}^{\mathrm{LDT}}) \\ \textit{verifier time} & \mathsf{t}_{\mathsf{V}} & = & O(\mathsf{t}_{\mathsf{V}}^{\mathcal{R}} + \mathsf{t}_{\mathsf{V}}^{\mathrm{LDT}}) \end{array}\right| ,$$

provided $\delta^{\mathrm{LDT}} < \min(\frac{1-2\rho_{\mathsf{c}}}{2}, \frac{1-\rho_{\mathsf{c}}}{3}, 1 - \rho_{\mathsf{e}})$*, and where* c *is the maximum size of a constraint set output by* $V_{\mathcal{R}}$*. (Parameters with superscript "*$\mathcal{R}$*" and "*LDT*" are parameters for* $(P_{\mathcal{R}}, V_{\mathcal{R}})$ *and* $(P_{\mathrm{LDT}}, V_{\mathrm{LDT}})$ *respectively.)*

Moreover, if $(P_{\mathcal{R}}, V_{\mathcal{R}})$ is an RS-encoded IOP of knowledge, then (P, V) is an IOP of knowledge.

Recall that $\ell_i^{\mathcal{R}}$ is the height of the i-th prover message, i.e., the i-th prover message has alphabet $\mathbb{F}^{\ell_i^{\mathcal{R}}}$.

Protocol 3.7.2. Letting $(P_{\mathcal{R}}, V_{\mathcal{R}})$ and $(P_{\mathrm{LDT}}, V_{\mathrm{LDT}})$ be as in the theorem statement, we need to construct an IOP (P, V) for $\mathcal{R}$. The prover P and verifier V both receive as input an instance $\mathbb{x}$, and the prover P also receives as input a corresponding witness $\mathbb{w}$.

1. **RS-encoded IOP for $\mathcal{R}$.** $\mathbf{P}$ and $\mathbf{V}$ simulate $(P_{\mathcal{R}}(\mathbb{x}, \mathbb{w}), V_{\mathcal{R}}(\mathbb{x}))$. During this protocol, the prover sends oracle codewords $\pi_1 \in \mathrm{RS}[L, \vec{\rho}_1], \ldots, \pi_{\mathsf{k}^{\mathcal{R}}} \in \mathrm{RS}[L, \vec{\rho}_{\mathsf{k}^{\mathcal{R}}}]$, and the verifier outputs a set of rational constraints $\mathfrak{C}$. Let $\ell := \sum_{i=1}^{\mathsf{k}^{\mathcal{R}}} \ell_i + |\mathfrak{C}|$, $\vec{\rho} = (\vec{\rho}_1, \ldots, \vec{\rho}_{\mathsf{k}^{\mathcal{R}}})$, $\vec{\rho}_{\mathfrak{C}} := (\sigma)_{(\mathcal{C},\sigma)\in\mathfrak{C}}$, and $\vec{\sigma} := (\vec{\rho}, \vec{\rho}_{\mathfrak{C}}) \in (0,1]^{\ell}$.
2. **Random linear combination.** $\mathbf{V}$ samples $\vec{z} \in \mathbb{F}^{2\ell}$ uniformly at random and sends it to $\mathbf{P}$.
3. **Low-degree test.** $\mathbf{P}$ and $\mathbf{V}$ simulate $(P_{\mathrm{LDT}}(\vec{z}^{\top}\Pi), V_{\mathrm{LDT}}^{\vec{z}^{\top}\Pi})$ where $\Pi := \left[\begin{smallmatrix}\Pi_0\\ \Pi_1\end{smallmatrix}\right] \in \mathbb{F}^{2\ell\times L}$ is as follows:
   - $\Pi_0' \in \mathbb{F}^{\ell\times L}$ is the matrix obtained by "stacking" vertically the matrices $\pi_1, \ldots, \pi_{\mathsf{k}^{\mathcal{R}}}$, and Π_0 is obtained by stacking Π_0' with $(\mathcal{C}[\Pi_0'])_{(\mathcal{C},\sigma)\in\mathfrak{C}}$.
   - $\Pi_1 \in \mathbb{F}^{\ell\times L}$ is the matrix whose entries are $(\Pi_1)_{i,a} := a^{(\rho_{\mathsf{c}}-\sigma_i)|L|} \cdot (\Pi_0)_{i,a}$ for all $i \in \{1, \ldots, \ell\}, a \in L$.
4. V accepts if and only if V_{LDT} accepts.

Proof.

Completeness. If $\pi_i \in \mathrm{RS}[L, \vec{\rho}_i]$ for all i, then $\vec{z}^{\top}\Pi \in \mathrm{RS}[L, \rho_{\mathsf{c}}]$ and thus P_{LDT} makes V_{LDT} accept. Completeness then follows immediately from the completeness of the RS-encoded IOP $(P_{\mathcal{R}}, V_{\mathcal{R}})$.

Soundness. Suppose that $\mathbb{x} \notin \mathcal{L}(\mathcal{R})$ and fix a malicious prover; let $\delta := \delta^{\mathrm{LDT}}$. During the protocol, the prover sends oracles $\tilde{\pi}_1, \ldots, \tilde{\pi}_{\mathsf{k}^{\mathcal{R}}}$; let $\tilde{\Pi} := \left[\begin{smallmatrix}\tilde{\Pi}_0\\ \tilde{\Pi}_1\end{smallmatrix}\right]$ be as in the protocol description but with respect to the messages $\tilde{\pi}_i$. We argue that the verifier accepts with probability at most $\varepsilon^{\mathcal{R}} + |L|/|\mathbb{F}| + \varepsilon_{\mathsf{i}}^{\mathrm{LDT}}$. To do this, we first show that it must hold that $\Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) > \delta$; then we show that given this, the verifier's acceptance probability is bounded as required. Recall that here Δ denotes the column-wise distance (see Section 2.1.1).

Let E be the event that the verifier accepts in the query phase with probability greater than $\varepsilon_{\mathsf{q}}^{\mathrm{LDT}}$, given the transcript of the interactive phase. Observe that

$$\begin{aligned}\Pr[E] &= \Pr[E \mid \Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) > \delta] \cdot \Pr[\Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) > \delta] \\ &\quad + \Pr[E \mid \Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) \le \delta] \cdot \Pr[\Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) \le \delta] \\ &\le \Pr[E \mid \Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) > \delta] + \Pr[\Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) \le \delta] \ .\end{aligned}$$

We will bound each of these terms individually. First, recall from Section 2.3.3 the properties of ρ_{e} and ρ_{c}.

- ρ_{c} is (at least) the maximum among the rates in $\vec{\rho} := (\vec{\rho}_i)_{i=1}^{\mathsf{k}}$ and the rates in $\{\sigma \ : \ \mathfrak{C} \in V \ , \ (\mathcal{C}, \sigma) \in \mathfrak{C}\}$;

- ρ_{e} is (at least) the maximum among ρ_{c} and the rates in $\{\mathrm{rate}(N;\vec{\rho}), \sigma + \mathrm{rate}(D) : \mathfrak{C} \in V\ ,\ (\mathcal{C},\sigma) \in \mathfrak{C}\}$.

In particular, $\rho_{\mathsf{e}} \geq \rho_{\mathsf{c}}$.

- **The probability of E when $\Delta(\tilde{\Pi}_0, \mathrm{RS}\left[L, \vec{\sigma}\right]) > \delta$.** First we argue that if $\Delta(\tilde{\Pi}_0, \mathrm{RS}\left[L, \vec{\sigma}\right]) > \delta$ then $\Delta(\tilde{\Pi}, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]^{2\ell}) > \delta$; then we cite a claim stating that, given this, a random linear combination of $\tilde{\Pi}$ is δ-far from $\mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]$ with high probability; finally we derive the bound of the aforementioned probability.

  Claim 3.7.3. *For any $\delta < (1-2\rho_{\mathsf{c}})/2$, if $\Delta(\tilde{\Pi}_0, \mathrm{RS}\left[L, \vec{\sigma}\right]) > \delta$ then $\Delta(\tilde{\Pi}, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]^{2\ell}) > \delta$.*

  Proof. Suppose by way of contradiction that $\Delta(\tilde{\Pi}, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]^{2\ell}) \leq \delta$, and let $\hat{\Pi} =: \left[\begin{smallmatrix}\hat{\Pi}_0\\ \hat{\Pi}_1\end{smallmatrix}\right] \in \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]^{\ell}$ be such that $\Delta(\tilde{\Pi}, \hat{\Pi}) \leq \delta$. For each i, let $p_i, p_i' \in \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]$ be the i-th rows of $\hat{\Pi}_0, \hat{\Pi}_1$ respectively. We argue that $p_i \in \mathrm{RS}\left[L, \sigma_i\right]$ for every i, which implies $\hat{\Pi}_0 \in \mathrm{RS}\left[L, \vec{\sigma}\right]$, so $\Delta(\tilde{\Pi}_0, \mathrm{RS}\left[L, \vec{\sigma}\right]) \leq \Delta(\tilde{\Pi}_0, \hat{\Pi}_0) \leq \Delta(\tilde{\Pi}, \hat{\Pi}) \leq \delta$, which is a contradiction.

  Suppose towards contradiction that there exists i such that $p_i \in \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right] \setminus \mathrm{RS}\left[L, \sigma_i\right]$. Then $q := p_i \cdot X^{(\rho_{\mathsf{c}}-\sigma_i)|L|} \in \mathrm{RS}\left[L, 2\rho_{\mathsf{c}} - \sigma_i\right] \setminus \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]$; in particular, $q \neq p_i'$, which implies that $\Delta(q, p_i') \geq 1 - (2\rho_{\mathsf{c}} - \sigma_i)$. However, because $\Delta(\tilde{\Pi}, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]^{2\ell}) \leq \delta$, we have that, letting $\tilde{p}_i$ be the i-th row of $\tilde{\Pi}_0$, $\Delta(\tilde{p}_i \cdot X^{(\rho_{\mathsf{c}}-\sigma_i)|L|}, q) = \Delta(\tilde{p}_i, p_i) \leq \delta$ and $\Delta(\tilde{p}_i \cdot X^{(\rho_{\mathsf{c}}-\sigma_i)|L|}, p_i') \leq \delta$. By the triangle inequality we have that $\Delta(q, p_i') \leq 2\delta < 1 - (2\rho_{\mathsf{c}} - \sigma_i)$, which is a contradiction. □

  Claim 3.7.4. *For all $\delta = d/|L| < (1-\rho_{\mathsf{c}})/3$, if $\Delta(\tilde{\Pi}, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]^{2\ell}) > \delta$ then*

$$\Pr_{\vec{z} \leftarrow \mathbb{F}^{2\ell}}\left[\Delta(\vec{z}^{\top}\tilde{\Pi}, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]) \geq \delta\right] \geq 1 - \frac{|L|}{|\mathbb{F}|}\ .$$

  Proof. It suffices to show that there *exists* $\vec{z}_0$ such that $\Delta(\vec{z}_0^{\top}\tilde{\Pi}, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]) > \delta$. The claim then follows from [39, Theorem 4.1] by setting $\epsilon := (|L|+|\mathbb{F}|^{-2\ell})^{-1}$, since $\Delta(\vec{z}^{\top}\tilde{\Pi}, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right])$ is an integer multiple of $|L|^{-1}$ and the above probability is an integer multiple of $|\mathbb{F}|^{-2\ell}$.

  If there exists a row $\tilde{\pi}_i$ of $\tilde{\Pi}$ such that $\Delta(\tilde{\pi}_i, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]) > \delta$ then we are done. So we assume that all rows are at a distance at most δ from $\mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]$.

  We may assume without loss of generality that the closest codeword to each row of $\tilde{\Pi}$ is zero, since distance is preserved under shifting by codewords. By assumption the number of nonzero columns of $\tilde{\Pi}$ is more than d. The probability that the inner product of a nonzero column with $\vec{z}$ is zero is $1/|\mathbb{F}|$, so by a union bound the probability that the inner product of any nonzero column with $\vec{z}$ is zero is at most $|L|/|\mathbb{F}|$; hence with probability at least $1 - |L|/|\mathbb{F}|$ it holds that $\Delta(\vec{z}^{\top}\tilde{\Pi}, 0) > \delta$. Fix some $\vec{z}$ such that this inequality holds. We show that there exists $\vec{z}_0$ such that $\delta < \Delta(\vec{z}_0^{\top}\tilde{\Pi}, 0) \leq 2\delta < 2(1-\rho_{\mathsf{c}})/3$; this implies that $\Delta(\vec{z}_0^{\top}\tilde{\Pi}, \mathrm{RS}\left[L, \rho_{\mathsf{c}}\right]) > \delta$ by the triangle inequality and the distance property of the code.

  For each $i \in [2\ell]$, let $p_i := \sum_{j=1}^{i} z_j \tilde{\pi}_j$; then $p_0 = 0$ and $p_{2\ell} = \vec{z}^{\top}\tilde{\Pi}$. We therefore have that $\Delta(p_0, 0) = 0$ and $\Delta(p_{2\ell}, 0) \geq \delta$. We also have that $\Delta(p_i, p_{i+1}) \leq \delta$, and so there exists i^* such that $\delta < \Delta(p_{i^*}, 0) \leq 2\delta$. We choose $\vec{z}_0$ to be the vector of the first i^* entries of $\vec{z}$ followed by zeroes. □

Combining the two claims: if $\Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) > \delta$ then, with probability at least $1 - |L|/|\mathbb{F}|$, it holds that $\Delta(\vec{z}^\top \tilde{\Pi}, \mathrm{RS}[L, \rho_{\mathsf{c}}]) \geq \delta$. Then $\Pr[E \mid \Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) > \delta] \leq \varepsilon_{\mathsf{i}}^{\mathrm{LDT}} + |L|/|\mathbb{F}|$ by definition of $\varepsilon_{\mathsf{q}}^{\mathrm{LDT}}$.

- **The probability that $\Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) \leq \delta$.** Let $\pi_1 \in \mathrm{RS}[L, \vec{\rho_1}], \ldots, \pi_{\mathsf{k}^{\mathcal{R}}} \in \mathrm{RS}[L, \vec{\rho}_{\mathsf{k}^{\mathcal{R}}}]$ be the closest codewords to the prover's messages $\tilde{\pi}_1, \ldots, \tilde{\pi}_{\mathsf{k}^{\mathcal{R}}}$. We can construct a prover $\hat{P}$ for the encoded IOP, which sends messages $\pi_1, \ldots, \pi_{\mathsf{k}^{\mathcal{R}}}$. We show that if $\Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) \leq \delta$, then for all $(\mathcal{C}, \sigma) \in \mathfrak{C}$ it holds that $\mathcal{C}[\hat{\Pi}'_0] \in \mathrm{RS}[L, \sigma]$. By the soundness of the encoded IOP, this occurs with probability at most $\varepsilon^{\mathcal{R}}$.

  Take some $(\mathcal{C}, \sigma) \in \mathfrak{C}$, and let $\pi_{\mathcal{C}} \in \mathrm{RS}[L, \sigma]$ be the (unique) closest codeword to $\mathcal{C}[\tilde{\Pi}'_0]$. Since $\Delta(\tilde{\Pi}'_0, \mathrm{RS}[L, \vec{\rho}]) \leq \delta$ and $\mathcal{C}$ operates column-wise, we have that $\Delta(\pi_{\mathcal{C}}, \mathcal{C}[\hat{\Pi}'_0]) \leq \delta$.

  Let $(N, D) := \mathcal{C}$; then $\Delta(\pi_{\mathcal{C}} \cdot D, N[\hat{\Pi}'_0]) \leq \delta$. Since $\pi_{\mathcal{C}} \cdot D \in \mathrm{RS}[L, \sigma + \mathrm{rate}(D)]$ and $N[\hat{\Pi}'_0] \in \mathrm{RS}[L, \mathrm{rate}(N; \vec{\rho})]$, we have that $\pi_{\mathcal{C}} \cdot D \equiv N[\hat{\Pi}'_0]$ since $\delta < 1 - \rho_{\mathsf{e}} \leq 1 - \max(\mathrm{rate}(N; \vec{\rho}), \sigma + \mathrm{rate}(D))$. In particular, this implies that D divides $N[\hat{\Pi}'_0]$ as a polynomial, and so $\mathcal{C}[\hat{\Pi}'_0] \in \mathrm{RS}[L, \mathrm{rate}(N; \vec{\rho}) - \mathrm{rate}(D)]$. (Note that $\perp \notin \mathcal{C}[\hat{\Pi}'_0]$ because otherwise the completeness condition of the RS-encoded IOP would fail to hold.) We conclude that $\mathcal{C}[\hat{\Pi}'_0] = \pi_{\mathcal{C}} \in \mathrm{RS}[L, \sigma]$.

Proof of knowledge. Suppose that $\tilde{P}$ causes the verifier to accept with probability μ. By a union bound, the probability that $\Delta(\tilde{\Pi}_0, \mathrm{RS}[L, \vec{\sigma}]) \leq \delta$ is at least $\mu - (|L|/|\mathbb{F}| + \varepsilon_{\mathsf{i}}^{\mathrm{LDT}} + \varepsilon_{\mathsf{q}}^{\mathrm{LDT}})$. Using an efficient algorithm for Reed–Solomon decoding (such as Berlekamp–Welch), we can recover $\hat{\Pi}'_0$ from $\tilde{\Pi}'_0$. By the above soundness analysis, we have that for all $(\mathcal{C}, \sigma) \in \mathfrak{C}$ we have $\mathcal{C}[\hat{\Pi}'_0] \in \mathrm{RS}[L, \sigma]$. The extractor runs $\tilde{P}$, corrects its messages and forwards them to the extractor for the RS-encoded IOP. By the knowledge guarantee of the RS-encoded IOP, the extractor succeeds with probability at least $\mu - (|L|/|\mathbb{F}| + \varepsilon_{\mathsf{i}}^{\mathrm{LDT}} + \varepsilon_{\mathsf{q}}^{\mathrm{LDT}} + \varepsilon_{\mathcal{R}})$. □

3.7.1 Zero knowledge

We describe how to modify the transformation above to preserve zero knowledge, thereby showing how to efficiently convert an RS-encoded IOP with a zero knowledge guarantee into a corresponding IOP with the same zero knowledge guarantee. The transformation uses the random self-reducibility of Reed–Solomon proximity testing, which implies that the low-degree test used in the transformation need not be zero knowledge (the only requirement is that its honest prover must run in polynomial time). In particular, the honest prover in the new protocol will send, in addition to the messages of the underlying RS-encoded IOP, a random codeword r, which is added to the linear combination of messages that are tested for proximity to RS.

Theorem 3.7.5. *Suppose that we are given:*

- *an RS-encoded IOP $(P_{\mathcal{R}}, V_{\mathcal{R}}, (\vec{\rho}_i)_{i=1}^{\mathsf{k}^{\mathcal{R}}})$, with maximum rate $(\rho_{\mathsf{c}}, \rho_{\mathsf{e}})$, for a relation $\mathcal{R}$ that is zero knowledge against b queries;*
- *an IOPP $(P_{\mathrm{LDT}}, V_{\mathrm{LDT}})$ for the RS code $\mathrm{RS}[L, \rho_{\mathsf{c}}]$ with a polynomial-time honest prover (not necessarily zero knowledge).*

Then we can combine these two ingredients to obtain an IOP (P, V) for $\mathcal{R}$, also zero knowledge against b queries, with the following parameters:

alphabet	Σ	$=$	$\Sigma^{\mathcal{R}}$
number of rounds	k	$=$	$\mathsf{k}^{\mathcal{R}} + \mathsf{k}^{\mathrm{LDT}} + 1$
proof length	p	$=$	$\mathsf{p}^{\mathcal{R}} + \mathsf{p}^{\mathrm{LDT}} + \lvert L \rvert$
query complexity	q_π	$=$	$\mathsf{q}_\pi^{\mathrm{LDT}} + \mathsf{q}_{\mathsf{w}}^{\mathrm{LDT}} \cdot \sum_{i=1}^{\mathsf{k}} \ell_i^{\mathcal{R}}$
randomness	$(\mathsf{r}_{\mathsf{i}}, \mathsf{r}_{\mathsf{q}})$	$=$	$\left(\mathsf{r}_{\mathsf{i}}^{\mathcal{R}} + \mathsf{r}_{\mathsf{i}}^{\mathrm{LDT}} + (\sum_{i=1}^{\mathsf{k}} \ell_i^{\mathcal{R}} + c) \log \lvert\mathbb{F}\rvert, \mathsf{r}_{\mathsf{q}}^{\mathrm{LDT}}\right)$
soundness error	$(\varepsilon_{\mathsf{i}}, \varepsilon_{\mathsf{q}})$	$=$	$\left(\varepsilon^{\mathcal{R}} + \frac{\lvert L \rvert}{\lvert\mathbb{F}\rvert} + \varepsilon_{\mathsf{i}}^{\mathrm{LDT}}, \varepsilon_{\mathsf{q}}^{\mathrm{LDT}}\right)$
prover time	t_{P}	$=$	$O(\mathsf{t}_{\mathsf{P}}^{\mathcal{R}} + \mathsf{t}_{\mathsf{P}}^{\mathrm{LDT}})$
verifier time	t_{V}	$=$	$O(\mathsf{t}_{\mathsf{V}}^{\mathcal{R}} + \mathsf{t}_{\mathsf{V}}^{\mathrm{LDT}})$

provided $\delta^{\mathrm{LDT}} < \min(\frac{1-2\rho_{\mathsf{c}}}{2}, \frac{1-\rho_{\mathsf{c}}}{3}, 1-\rho_{\mathsf{e}})$, and where c is the maximum size of a constraint set output by $V_{\mathcal{R}}$. (Parameters with superscript "$\mathcal{R}$" and "LDT" are parameters for $(P_{\mathcal{R}}, V_{\mathcal{R}})$ and $(P_{\mathrm{LDT}}, V_{\mathrm{LDT}})$ respectively; highlights denote parameter differences with Theorem 3.7.1.)

Protocol 3.7.6. Letting $(P_{\mathcal{R}}, V_{\mathcal{R}})$ and $(P_{\mathrm{LDT}}, V_{\mathrm{LDT}})$ be as in the theorem statement, we need to construct an IOP (P, V) for $\mathcal{R}$. The prover P and verifier V both receive as input an instance $\mathbb{x}$, and the prover P also receives as input a corresponding witness $\mathbb{w}$.

1. **Masking codeword for low-degree test.** $\mathbf{P}$ sends to $\mathbf{V}$ a random $r \in \mathrm{RS}[L, \rho_{\mathsf{c}}]$.
2. **RS-encoded IOP for $\mathcal{R}$.** In parallel to the above, $\mathbf{P}$ and $\mathbf{V}$ simulate $(P_{\mathcal{R}}(\mathbb{x}, \mathbb{w}), V_{\mathcal{R}}(\mathbb{x}))$. In the course of this protocol, the prover sends oracle codewords $\pi_1 \in \mathrm{RS}[L, \vec{\rho}_1], \ldots, \pi_{\mathsf{k}^{\mathcal{R}}} \in \mathrm{RS}[L, \vec{\rho}_{\mathsf{k}^{\mathcal{R}}}]$, and the verifier specifies a set of rational constraints $\mathfrak{C}$. Let $\ell := \sum_{i=1}^{\mathsf{k}^{\mathcal{R}}} \ell_i + \lvert\mathfrak{C}\rvert$.
3. **Random linear combination.** $\mathbf{V}$ samples $\vec{z} \in \mathbb{F}^{2\ell}$ uniformly at random and sends it to $\mathbf{P}$.
4. **Low-degree test.** $\mathbf{P}$ and $\mathbf{V}$ simulate $(P_{\mathrm{LDT}}(\vec{z}^{\top}\Pi + r), V_{\mathrm{LDT}}^{\vec{z}^{\top}\Pi + r})$ where $\Pi := \left[\begin{smallmatrix}\Pi_0\\ \Pi_1\end{smallmatrix}\right] \in \mathbb{F}^{2\ell \times L}$ is defined as in Protocol 3.7.2.
5. V accepts if only if V_{LDT} accepts.

Proof. Completeness and soundness follow almost immediately from those of Protocol 3.7.2. Indeed, we can view Protocol 3.7.6 as Protocol 3.7.2 modified so that $(P_{\mathcal{R}}, V_{\mathcal{R}})$ begins with an additional "dummy" round where the prover just sends a random codeword. (Note that we can fix $\vec{z}$'s random coefficient for r to be 1 almost without loss of generality since distance to Reed–Solomon codewords is preserved under multiplication by a nonzero constant.) We now focus on arguing the zero knowledge property.

Let $S_{\mathcal{R}}$ be the simulator for $(P_{\mathcal{R}}, V_{\mathcal{R}})$, witnessing zero knowledge against b queries. The simulation guarantee for $S_{\mathcal{R}}$ is that, for any $\tilde{V}_{\mathcal{R}}$ that makes at most b distinct queries across all oracles, $\mathrm{View}(P_{\mathcal{R}}(\mathbb{x}, \mathbb{w}), \tilde{V}_{\mathcal{R}})$ and the output of $S_{\mathcal{R}}^{\tilde{V}_{\mathcal{R}}}(\mathbb{x})$ are identically distributed.

Consider the simulator S for (P, V) that, given a malicious verifier $\tilde{V}$, constructs a new malicious verifier $\tilde{V}_{\mathcal{R}}$ (defined below), then runs $S_{\mathcal{R}}$ on $\tilde{V}_{\mathcal{R}}$, and finally outputs what $\tilde{V}_{\mathcal{R}}$ outputs given its simulated view.

1. Start running $\tilde{V}$.
2. Sample $r_{\mathrm{sim}} \in \mathrm{RS}[L, \rho_{\mathsf{c}}]$ uniformly at random, and answer $\tilde{V}$'s queries to r with r_{sim}; let Q_{sim} be the verifier's queries to r in this phase.

3. For $\mathsf{k}^{\mathcal{R}}$ rounds, forward $\tilde{V}$'s messages to the prover. Answer all of $\tilde{V}$'s queries to the received oracles honestly. Receive a set of rational constraints $\tilde{\mathfrak{C}}$ from $\tilde{V}$.
4. Receive $\tilde{z} \in \mathbb{F}^{2\ell}$ from $\tilde{V}$.
5. For every $\omega \in Q_{\mathrm{sim}}$, query every oracle received at ω. For each oracle defined by a rational constraint $(\tilde{\mathcal{C}}, \tilde{\sigma}) \in \tilde{\mathfrak{C}}$, evaluate $\tilde{\mathcal{C}}$ at ω.
6. Let $p_0^{(\omega)} \in \mathbb{F}^{\ell}$ be the value of each oracle at point ω, $p_1^{(\omega)}$ be given by $(p_1^{(\omega)})_i := \omega^{\rho-\sigma_i}(p_0^{(\omega)})_i$ for $i \in [\ell]$, and $p^{(\omega)} \in \mathbb{F}^{2\ell}$ be the concatenation of $p_0^{(\omega)}$ and $p_1^{(\omega)}$.
7. Sample $p_{\mathrm{sim}} \in \mathrm{RS}[L, \rho_{\mathsf{c}}]$ uniformly at random such that, for every $\omega \in Q_{\mathrm{sim}}$, $p_{\mathrm{sim}}(\omega) = \tilde{z}^{\top} p^{(\omega)} + r_{\mathrm{sim}}(\omega)$.
8. Now when $\tilde{V}$ queries r at ω, query every oracle received at ω and answer with $p_{\mathrm{sim}}(\omega) - \tilde{z}^{\top} p^{(\omega)}$.
9. Simulate the interaction of $P_{\mathrm{LDT}}(p)$ and $\tilde{V}$.
10. Output the view of the simulated $\tilde{V}$.

For every query that $\tilde{V}$ makes to r, $\tilde{V}_{\mathcal{R}}$ makes a query to every oracle it has received in the same location. Similarly, for each query $\tilde{V}$ makes to any other oracle, $\tilde{V}_{\mathcal{R}}$ makes at most one query to some received oracle. Hence $\tilde{V}_{\mathcal{R}}$ makes at most b distinct queries across all oracles, and so the simulation guarantee holds.

To show zero knowledge, we exhibit the following hybrid experiment, in which the view of $\tilde{V}$ is identically distributed to the output of the simulator.

1. Run the honest prover $P_{\mathcal{R}}(\mathbb{x}, f)$; let Π be as in the protocol.
2. Sample $r \in \mathrm{RS}[L, \rho_{\mathsf{c}}]$ uniformly at random and send it to $\tilde{V}$. Let $Q \subseteq L$ be the verifier's queries to r in this phase.
3. Receive $\tilde{z} \in \mathbb{F}^{2\ell}$ from $\tilde{V}$.
4. Sample $p \in \mathrm{RS}[L, \rho_{\mathsf{c}}]$ uniformly at random such that, for every $\omega \in Q$, $p(\omega) = (\tilde{z}^{\top}\Pi)_{\omega} + r(\omega)$.
5. Replace r with $p - \tilde{z}^{\top}\Pi$.
6. Simulate the interaction of $P_{\mathrm{LDT}}(p)$ and $\tilde{V}$.

One can verify that the view of $\tilde{V}$ in this hybrid is also identically distributed to the view of $\tilde{V}$ in the real protocol. In particular, all answers to $\tilde{V}$'s queries to r after its replacement by p are correctly distributed. □

3.8 Aurora: an IOP for R1CS

We describe the IOP for R1CS (Definition 3.6.1) that comprises the main technical contribution of this paper, and also underlies the zkSNARK for R1CS that we have designed and built (more about this in Section 3.9).

For the discussions below, we introduce notation about the low-degree test in [22], known as "Fast Reed–Solomon IOPP" (FRI): given a subspace L of a binary field $\mathbb{F}$ and rate $\rho \in (0, 1)$, we denote by $\varepsilon_{\mathsf{i}}^{\mathrm{FRI}}(\mathbb{F}, L)$ and $\varepsilon_{\mathsf{q}}^{\mathrm{FRI}}(L, \rho, \delta)$ the soundness error of the interactive and query phases

in FRI (respectively) when testing proximity of a δ-far function to $\mathrm{RS}[L, \rho]$. See Appendix A.3.1 for details about these functions.

We first provide a "barebones" statement with constant soundness error and no zero knowledge.

Theorem 3.8.1. *There is an IOP of knowledge for $\mathcal{R}_{\mathrm{R1CS}}$ (Definition 3.6.1) over binary fields $\mathbb{F}$ that, given an R1CS instance having n variables and m constraints, letting $\rho \in (0,1)$ be a constant and L be any subspace of $\mathbb{F}$ such that $2\max(m, n+1) \leq \rho|L|$, has the following parameters:*

alphabet	Σ	$=$	$\mathbb{F}$
number of rounds	k	$=$	$O(\log \lvert L\rvert)$
proof length	p	$=$	$(5+\frac{1}{3})\lvert L\rvert$
query complexity	q_π	$=$	$O(\log \lvert L\rvert)$
randomness	$(\mathsf{r}_\mathsf{i}, \mathsf{r}_\mathsf{q})$	$=$	$(O(\log \lvert L\rvert \cdot \log \lvert\mathbb{F}\rvert), O(\log \lvert L\rvert))$
soundness error	$(\varepsilon_\mathsf{i}, \varepsilon_\mathsf{q})$	$=$	$\left(\frac{m+1}{\lvert\mathbb{F}\rvert} + \frac{\lvert L\rvert}{\lvert\mathbb{F}\rvert} + \varepsilon_\mathsf{i}^{\mathrm{FRI}}(\mathbb{F}, L), \varepsilon_\mathsf{q}^{\mathrm{FRI}}(L, \rho, \delta)\right)$
prover time	t_P	$=$	$O(\lvert L\rvert \cdot \log(n+m) + \lVert A\rVert + \lVert B\rVert + \lVert C\rVert) + 17 \cdot \mathrm{FFT}(\mathbb{F}, \lvert L\rvert)$
verifier time	t_V	$=$	$O(\lVert A\rVert + \lVert B\rVert + \lVert C\rVert + n + m + \log \lvert L\rvert)$

where $\delta := \min(\frac{1-2(\rho/2)}{2}, \frac{1-(\rho/2)}{3}, 1-\rho)$.

Proof. Apply the transformation in Section 3.7 (see Theorem 3.7.1) to two ingredients: (a) the RS-encoded IOP for R1CS in Section 3.6 (see Theorem 3.6.2); and (b) the FRI low-degree test with proximity parameter $\delta^{\mathrm{LDT}} := \delta$. Note that the condition on δ^{LDT} is satisfied by definition. The resulting protocol is sound against *all* malicious provers (and not just provers that send oracles that are Reed–Solomon codewords). □

Next, we provide a statement that additionally has parameters for controlling the soundness error, is zero knowledge, and includes other (whitebox) optimizations; the proof is analogous except that we use zero knowledge components (the RS-encoded IOP of Theorem 3.6.4 and the transformation of Theorem 3.7.5). The resulting IOP protocol, fully specified in Fig. 3.3, underlies our zkSNARK for R1CS (see Section 3.9).

Theorem 3.8.2. *There is an IOP of knowledge for $\mathcal{R}_{\mathrm{R1CS}}$ (Definition 3.6.1) over binary fields $\mathbb{F}$ that, given an R1CS instance having n variables and m constraints, letting $\rho \in (0,1)$ be a constant and L be any subspace of $\mathbb{F}$ such that $2\max(m, n+1) + 2\mathsf{b} \leq \rho|L|$, is zero knowledge against b queries and has the following parameters:*

alphabet	Σ	$=$	$\mathbb{F}$
number of rounds	k	$=$	$O(\log \lvert L\rvert)$
proof length	p	$=$	$(4 + 2\lambda_\mathsf{i} + \lambda'_\mathsf{i}\lambda_\mathsf{i}^{\mathrm{FRI}}/3)\lvert L\rvert$
query complexity	q_π	$=$	$O(\lambda_\mathsf{i}\lambda_\mathsf{i}^{\mathrm{FRI}}\lambda_\mathsf{q}^{\mathrm{FRI}} \log \lvert L\rvert)$
randomness	$(\mathsf{r}_\mathsf{i}, \mathsf{r}_\mathsf{q})$	$=$	$\big(O((\lambda_\mathsf{i}\lambda'_\mathsf{i} + \lambda_\mathsf{i}^{\mathrm{FRI}} \log \lvert L\rvert) \log \lvert\mathbb{F}\rvert), O(\lambda_\mathsf{q}^{\mathrm{FRI}} \log \lvert L\rvert)\big)$
soundness error	$(\varepsilon_\mathsf{i}, \varepsilon_\mathsf{q})$	$=$	$\left((\frac{m+1}{\lvert\mathbb{F}\rvert})^{\lambda_\mathsf{i}} + (\frac{\lvert L\rvert}{\lvert\mathbb{F}\rvert})^{\lambda'_\mathsf{i}} + \varepsilon_\mathsf{i}^{\mathrm{FRI}}(\mathbb{F}, L)^{\lambda_\mathsf{i}^{\mathrm{FRI}}}, \varepsilon_\mathsf{q}^{\mathrm{FRI}}(L, \rho, \delta)^{\lambda_\mathsf{q}^{\mathrm{FRI}}}\right)$
prover time	t_P	$=$	$\lambda_\mathsf{i} \cdot (O(\lvert L\rvert \cdot \log(n+m) + \lVert A\rVert + \lVert B\rVert + \lVert C\rVert) + 18 \cdot \mathrm{FFT}(\mathbb{F}, \lvert L\rvert)) + O(\lambda'_\mathsf{i}\lambda_\mathsf{i}^{\mathrm{FRI}}\lvert L\rvert)$
verifier time	t_V	$=$	$\lambda_\mathsf{i} \cdot O(\lVert A\rVert + \lVert B\rVert + \lVert C\rVert + n + m + \log \lvert L\rvert) + O(\lambda'_\mathsf{i}\lambda_\mathsf{i}^{\mathrm{FRI}}\lambda_\mathsf{q}^{\mathrm{FRI}} \log \lvert L\rvert)$

where $\delta := \min(\frac{1-2\rho}{2}, \frac{1-\rho}{3}, 1-\rho)$. *Setting* $\mathsf{b} \geq \mathsf{q}_\pi$ *ensures honest-verifier zero knowledge.*

Given an R1CS instance (I, v), we fix subspaces $H_1, H_2 \subseteq \mathbb{F}$ such that $|H_1| = m$ and $|H_2| = n + 1$ (padding to the nearest power of 2 if necessary) with $H_1 \subseteq H_2$ or $H_2 \subseteq H_1$, and a sufficiently large affine subspace $L \subseteq \mathbb{F}$ such that $L \cap (H_1 \cup H_2) = \varnothing$. We let $t := |H_1 \cup H_2| = \max(m, n+1)$. Fig. 3.2 below gives polynomials and codewords used in Fig. 3.3. We also define $\xi := \sum_{a \in H_1 \cup H_2} a^{t-1}$.

Finally, we note that we have tailored the transformation from Section 3.7 to the RS-encoded IOP from Section 3.6 in the sense that for most oracles of sub-maximal rate it actually suffices to test proximity to the maximal rate so they only appear once (unshifted) in the matrix Π. The single exception is the (virtual) oracle g_i used in the sumcheck protocol, for which we *must* test proximity with rate exactly $(t-1)/|L|$. Because of this, g_i is the only oracle for which we test both g_i and a shift of it, as is shown in the matrix Π in Fig. 3.3.

polynomial	**degree**	**values that define the polynomial**
p_α	$t-1$	$\hat{p}_\alpha(a) = \begin{cases} \alpha^{\gamma(a)} & \text{for } a \in H_1 \\ 0 & \text{for } a \in (H_1 \cup H_2) \setminus H_1 \end{cases}$
$p_\alpha^{(M)}$	$t-1$	$\hat{p}_\alpha^{(M)}(b) = \begin{cases} \sum_{a \in H_1} M_{a,b} \cdot \alpha^{\gamma(a)} & \text{for } b \in H_2 \\ 0 & \text{for } b \in (H_1 \cup H_2) \setminus H_2 \end{cases}$
codeword	**code**	**polynomial that defines the codeword**
f_w	$\mathrm{RS}\left[L, \frac{n-k+\mathsf{b}}{\lvert L\rvert}\right]$	random polynomial $\bar{f}_w$ of degree less than $n-k+\mathsf{b}$ such that, for all $b \in H_2$ with $k < \gamma(b) \leq n$, $\bar{f}_w(b) = \dfrac{w_{\gamma(b)-k} - \hat{f}_{(1,v)}(b)}{\mathbb{Z}_{H_2^{\leq k}}(b)}$
f_{Mz}	$\mathrm{RS}\left[L, \frac{m+\mathsf{b}}{\lvert L\rvert}\right]$	random polynomial $\bar{f}_{Az}$ of degree less than $m+\mathsf{b}$ such that, for all $a \in H_1$, $\bar{f}_{Az}(a) = \sum_{b \in H_2} M_{a,b} \cdot z_{\gamma(b)} = (Mz)_a$

Figure 3.2: Polynomials and codewords used in the IOP protocol given in Fig. 3.3.

3.9 `libiop`: a library for IOP-based SNARGs

We provide `libiop` (see `https://github.com/scipr-lab/libiop`), a codebase that enables the design and implementation of IOP-based non-interactive arguments. The codebase uses the C++ language and has three main components: (1) a library for writing IOP protocols; (2) a realization of the [32] transformation, mapping any IOP written with our library to a corresponding non-interactive argument; (3) a portfolio of IOP protocols. We discuss each of these components in turn.

3.9.1 Library for IOP protocols

We provide a library that enables a programmer to write IOP protocols. Informally, the programmer provides a blueprint of the IOP by specifying, for each round, the number and

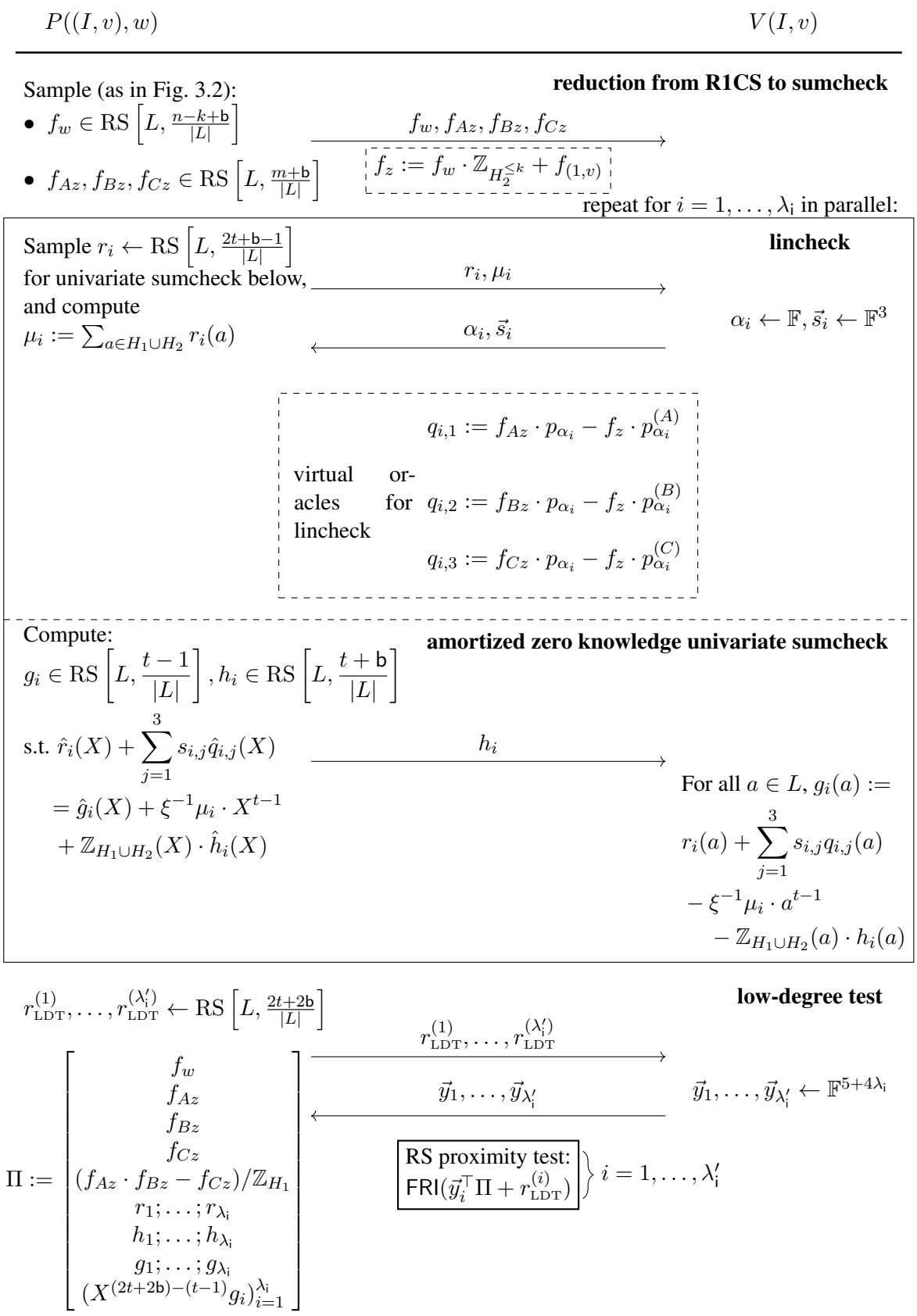


Figure 3.3: Diagram of the zero knowledge IOP for R1CS that proves Theorem 3.8.2.

sizes of oracle messages (and non-oracle messages) sent by the prover, as well as the number of random bytes subsequently sent by the verifier. For the prover, the programmer specifies how each message is to be computed. For the verifier, the programmer specifies how oracle queries are generated and, also, how the verifier's decision is computed based on its random choices and information received from the prover. Notable features of our library include:

- Support for writing new IOPs by using other IOPs as sub-protocols. This includes juxtaposing or interleaving selected rounds of these sub-protocols. This latter feature not only facilitates reducing round complexity in complex IOP constructions but also makes it possible to take advantage of optimizations such as column hashing (discussed in Section 3.9.2) when constructing a non-interactive argument.
- A realization of the transformation described in Section 3.7, which constructs an IOP by combining an encoded IOP (as defined in Section 2.3.3) and a low-degree test (as defined in Section 2.3.1.1). This is a powerful paradigm (it applies to essentially all published IOP protocols) that reduces the task of writing an IOP to merely providing suitable choices of these two simpler ingredients.

3.9.2 BCS transformation

We realize the transformation of [32], by providing code that maps any IOP written in our library into a corresponding non-interactive argument (which consists of a prover algorithm and a verifier algorithm).

We use BLAKE2b [14] to instantiate the random oracle in the [32] transformation (our code allows to conveniently specify alternative instantiations). This hash function is an improvement to BLAKE (a finalist in the SHA-3 competition) [13], and its performance on all recent x86 platforms is competitive with the most performant (and often hardware-accelerated) hash functions [122]. Moreover, BLAKE2b can be configured to output digests of any length between 1 and 64 bytes (between 8 and 512 bits in multiples of 8). When aiming for a security level of λ bits, we only need the hash function to output digests of 2λ bits, and our code automatically sets this length.

Our code incorporates additional optimizations that, while simple, are generic and effective.

One is *column hashing*, which informally works as follows. In many IOP protocols (essentially all published ones, including Ligero [9] and Stark [23]), the prover sends multiple oracles over the same domain in the same round, and the verifier accesses all of them at the same index in the domain. The prover can then build a Merkle tree over *columns* consisting of corresponding entries of the oracles, rather than building separate Merkle trees for each or a single Merkle tree over their concatenation. This reduces a non-interactive proof's length, because the proof only has to contain a *single* authentication path for the desired column, rather than authentication paths corresponding to the indices across all the oracles.

Another optimization is *path pruning*. When providing multiple authentication paths relative to the same root (in the non-interactive argument), some digests become redundant and can thus be omitted. For example, if one considers the authentication paths for all leaves in a particular sub-tree, then one can simple provide the authentication path for the root of the

sub-tree. A simple way to view this optimization is to provide the smallest number of digests to authenticate a *set* of leaves.

3.9.3 Portfolio of IOP protocols and sub-components

We have used our library to realize the IOP protocols below. (We have publicly released the first two.)

- **Aurora:** our IOP protocol for R1CS (specifically, the one provided in Fig. 3.3 in Section 3.8).
- **Ligero:** an adaptation of the IOP protocol in [9] to R1CS. While the protocol(s) in [9] are designed for (boolean or arithmetic) circuit satisfiability, the same ideas can be adapted to support R1CS *at no extra cost*. This simplifies comparisons with R1CS-based arguments, and confers additional expressivity.
- **Stark:** the IOP protocol in [23] for *Algebraic Placement and Routing* (APR), a language that is a "succinct" analogue of algebraic satisfaction problems such as R1CS. (See [23] for details.)

Each of these IOPs is obtained by specifying an encoded IOP and a low-degree test. As explained in Sections 3.9.1 and 3.9.2, our library compiles these into an IOP protocol, and the latter into a non-interactive argument. This toolchain enables writing protocols with fewer lines of code, and enhances code auditability.

The IOP protocols above benefit from several algebraic components that our library also provides.

- *Finite field arithmetic.* We support prime and binary fields. Our prime field arithmetic uses Montgomery representation [120]. Our binary field arithmetic uses the carryless multiplication instructions [98]; these are ubiquitous in x86 CPUs and, being used in AES-GCM computations, are highly optimized.
- *FFT algorithms.* The choice of FFT algorithm depends on whether the R1CS instance (and thus the rest of the protocol) is defined over a prime or binary field. In the former case, we use the radix-2 FFT (whose evaluation domain is a multiplicative coset of order 2^a for some a) [72]. In the latter case, we use an additive FFT (whose evaluation domain is an affine subspace of the binary field) [62, 86, 43, 110, 109]. We also provide the respective inverse FFTs, and variants for cosets of the base domains.

Remark 3.9.1. Known techniques can be used to reduce given programs or general machine computations to low-level representations such as R1CS and APR (see, e.g., [34, 147, 23]). Such techniques have been compared in prior work, and our library does not focus on these.

3.10 Evaluation

In Section 3.10.1 we evaluate the performance of Aurora. Then, in Section 3.10.2 we compare Aurora with Ligero [9] and Stark [23], two other IOP-based zkSNARKs. Our experiments

not only demonstrate that Aurora's performance matches the theoretical predictions implied by the protocol but also that Aurora achieves the smallest argument size of any IOP-based zkSNARK, *by more than an order of magnitude*.

That said, there is still a sizable gap between the argument sizes of IOP-based zkSNARKs and other zkSNARKs that use public-key cryptographic assumptions vulnerable to quantum adversaries; see Fig. 1.2 for how argument sizes vary across these. It remains an exciting open problem to close this gap.

Experiments ran on a machine with an Intel Xeon W-2155 3.30GHz 10-core processor and 64GB of RAM.

3.10.1 Performance of Aurora

We consider Aurora at the standard security level of 128 bits, over the binary field $\mathbb{F}_{2^{192}}$. We report data on key efficiency measures of a zkSNARK: the time to generate a proof (running time of the prover), the length of a proof, and the time to check a proof (running time of the verifier). We also indicate how much of each cost is due to the IOP protocol, and how much is due to the BCS transformation [32].

In Aurora, all of these quantities depend on the number of constraints m in an R1CS instance.[4] Our experiments report how these quantities change as we vary m over the range $\{2^{10}, 2^{11}, \ldots, 2^{20}\}$.

Prover running time. In Fig. 3.4 we plot the running time of the prover, as absolute cost (top left graph) and as relative cost when compared to native execution (bottom left graph). For R1CS, *native execution* is the time that it takes to check that an assignment satisfies the constraint system. The plot confirms the quasilinear complexity of the prover; proving times range from fractions of a second to several minutes. Proving time is dominated by the cost of running the underlying IOP prover.

Argument size. In Fig. 3.4 we plot argument size, as absolute cost (top middle graph) and as relative cost when compared to native witness size (bottom middle graph). For R1CS, *native witness size* is the number of bytes required to represent an assignment to the constraint system. The plot shows that compression (argument size is smaller than native witness size) occurs for $m \geq 2000$. The plot also shows that argument size ranges from $40\,\mathrm{kB}$ to $130\,\mathrm{kB}$, and is dominated by the cryptographic digests to authenticate query answers.

Verifier running time. In Fig. 3.4 we plot the running time of the verifier, as absolute cost (top right graph) and as relative cost when compared to native execution (bottom right graph). The plot shows that verification times range from milliseconds to seconds, and confirms that our implementation incurs a constant multiplicative overhead over native execution.

[4] The number of variables n also affects performance, but it is usually close to m and so we take $n \approx m$ in our experiments. The number of inputs k in an R1CS instance is at most n, and in typical applications it is much smaller than n, so we do not focus on it.

3.10.2 Comparison of Ligero, Stark, and Aurora

In Fig. 3.5 we compare costs (proving time, argument size, and verification time) on R1CS instances for three IOP-based zkSNARKs: Ligero [9], Stark [23], and Aurora (this work). As in Section 3.10.1, we plot costs as the number of constraints m increases (and with $n \approx m$ variables as explained in Footnote 4); we also set security to the standard level of 128 bits and use the binary field $\mathbb{F}_{2^{192}}$.

Comparison of Ligero and Aurora. Ligero natively supports R1CS so a comparison with Aurora is straightforward. Fig. 3.5 (middle graph) shows that argument size in Aurora is much smaller than in Ligero, even for a relatively small number of constraints. The gap between the two grows bigger as the number of constraints increases, as Aurora's argument size is polylogarithmic while Ligero's is only sublinear (an exponential gap).

Comparison of Stark and Aurora. Stark does not natively support the NP-complete relation R1CS but instead natively supports an NEXP-complete relation known as *Algebraic Placement and Routing* (APR). These two relations are quite different, and so to achieve a meaningful comparison, we consider an APR instance that *simulates* a given R1CS instance. We thus plot the costs of Stark on a hand-optimized APR instance that simulates R1CS instances. Relying on the reductions described in [23], we wrote an APR instance that realizes a simple abstract computer that checks that a variable assignment satisfies each one of the rank-1 constraints in a given R1CS instance.

Referring to Fig. 3.5, the middle graph shows that argument size in Aurora is much smaller than in Stark, even if both share the same asymptotic growth. This is due to the fact that R1CS and APR target different computation models (explicit circuits vs. uniform computations), so Stark incurs significant overheads when used for R1CS. The right graph shows that verification time in Stark grows linearly with the number of constraints (like Ligero and Aurora); indeed, the verifier must read the description of the statement being proved, which is the entire constraint system.

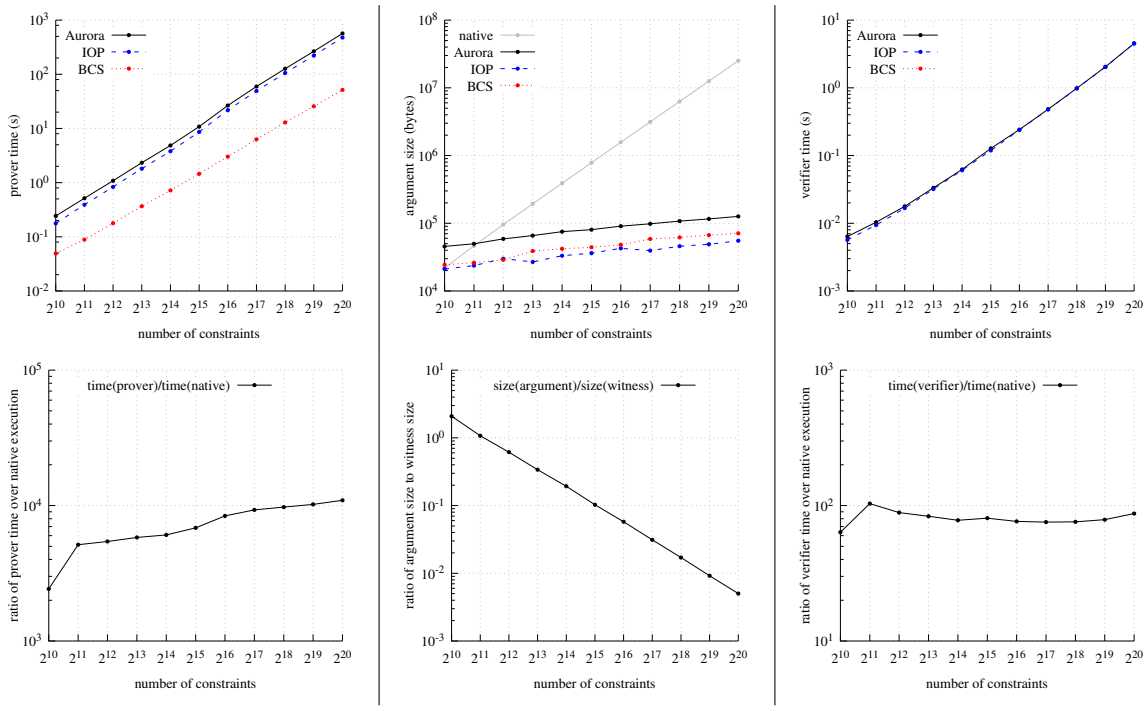


Figure 3.4: From left to right: proving time, argument size and verification time in Aurora. The top row of graphs shows absolute values; the bottom row shows the ratio to native execution.

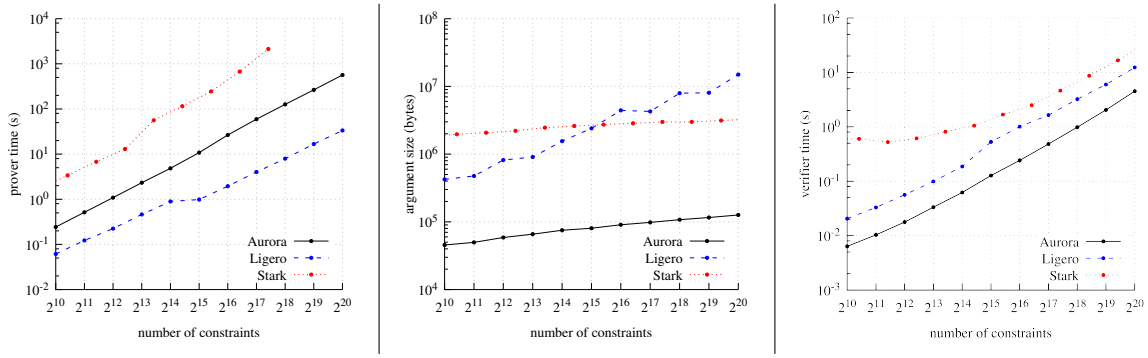


Figure 3.5: From left to right: comparisons of proving time, argument size and verification time in Aurora, Ligero and Stark.

Chapter 4

Linear-size IOPs for delegating computation

4.1 Introduction

Checking computations faster than they can be run is a central goal in the theory of computation. The study of proof protocols that enable fast verification has produced powerful tools for complexity theory and cryptography, and has even led to applications to real-world problems such as delegation of computation. For applications, it is crucial that the underlying complexity-theoretic objects are efficient.

An influential line of work began with *probabilistically checkable proofs* (PCPs) [16]. These are non-interactive proofs for membership in a language, which admit fast probabilistic verification based on local queries to the proof. While the most prominent application of PCPs is to hardness of approximation [82], seminal works of Kilian [107] and Micali [118] showed that PCPs can also be used to obtain computationally-sound schemes for delegation of computation that are asymptotically efficient.

The application of PCPs to delegation of computation singles out particular design objectives, distinct from those that arise from hardness of approximation. The relevant complexity measures for PCPs in the context of delegation are: *query complexity*, *verifier time*, and *prover time*. The latter two are self-explanatory, since the proof must be produced and validated; the former arises because, in existing delegation schemes based on PCPs, communication complexity depends linearly on the query complexity of the underlying PCP. Note that the running time of the prover is not typically considered in the context of PCPs, because one considers only the existence of a valid PCP string and not how it is constructed. For delegation schemes, on the other hand, the time required to generate the proof is often a barrier to practical use.

An ideal PCP for delegation would have *constant* query complexity, *(poly)logarithmic* verifier time, and *linear* prover time. This naturally implies that the *proof length* must also be linear, since it is a lower bound on the prover's running time in most applications. State-of-the-art PCPs achieve constant query complexity and polylogarithmic verifier time, but only *quasilinear* ($O(N \log^c N)$) proof length [119]. While the proof length is asymptotically close to optimal, c is a fairly large constant, and moreover the construction uses gap amplification techniques that are

not believed to be concretely efficient. The *only* construction of PCPs with linear proof length has query complexity $O(N^\epsilon)$ and a verifier that runs in non-uniform polynomial time [38]; the running time of the prover is not specified. Clearly, these parameters are a long way from those desired of PCPs for fast verification of computation.

In light of these apparent barriers, Ben-Sasson et al. [32] have demonstrated how to obtain computationally-sound delegation schemes from *interactive oracle proofs* (IOPs). This is a natural generalization of PCPs independently introduced by [32, 127] (also generalizing the "interactive PCP" model studied in [106]). An IOP is an interactive protocol consisting of multiple rounds, where in each round the verifier sends a challenge and the prover responds with a PCP oracle to which the verifier can make a small number of queries. The proof length of an IOP is the total length of all oracles sent by the prover, and the query complexity is the total number of queries made to these oracles. The study of IOPs explores the tradeoff between a new efficiency measure, round complexity, and other efficiency measures. Viewed in this way, a PCP is an IOP with optimal round complexity.

The work of [32] justifies why exploiting the tradeoff between round complexity and other efficiency measures is potentially advantageous for constructing computationally-sound delegation schemes. In particular, if we could obtain IOPs with constant round complexity that otherwise match the parameters of an ideal PCP (constant query complexity, polylogarithmic verifier time, linear prover time), then we would obtain delegation schemes that have the same asymptotic efficiency as those derived from an ideal PCP. Thus, for the purposes of verifiable delegation schemes, it suffices to construct such "ideal" IOPs.

A recent line of works has leveraged this tradeoff to establish a number of results that we do not know how to achieve via PCPs alone [26, 25, 21, 22, 23, 129]. Two constructions are particularly relevant for us: [25] achieves IOPs for Boolean circuit satisfiability (CSAT) with linear proof length, unspecified polynomial prover time, and constant query and round complexity,[1] and AURORA achieves logarithmic-round IOPs for arithmetic circuit satisfiability with $\tilde{O}(N \log N)$ prover time,[2] linear proof length, and logarithmic query complexity.

However, these IOPs do *not* have sublinear verifier time (see Section 4.1.2). The state-of-the-art for IOPs with polylogarithmic verifier time is [23], which achieves $O(N \log N)$ proof length, $\tilde{O}(N \log^2 N)$ prover time, and logarithmic query and round complexity. Our goal in this chapter is to make progress towards constructing ideal IOPs by giving a construction that simultaneously achieves the state of the art in all of these metrics: linear proof length, $\tilde{O}(N \log N)$ prover time, constant query and round complexity, and polylogarithmic verifier time.

4.1.1 Our results

In this chapter we construct IOPs for algebraic computations over large fields that are "almost" ideal; namely, we achieve linear proof length, $O(N \log N)$ (*strictly quasilinear*) prover arithmetic complexity, constant query and round complexity, and *polylogarithmic* verifier time. Our new IOP protocols match the state-of-the-art proof length and prover complexity of AURORA,

[1] Subsequent to this work, [129] showed how to achieve such IOPs for CSAT where the proof length is $(1 + \epsilon)N$ for any $\epsilon > 0$. Their verifier runs in time $\tilde{O}(N)$.

[2] Here the $\tilde{O}$ notation hides $\mathrm{poly}(\log \log N)$ factors.

while at the same time achieving an *exponential improvement* in verifier time for a rich class of computations. We focus on arithmetic complexity as the natural notion of efficiency for IOPs for algebraic problems.[3]

The arithmetic complexity of our prover is tightly connected to the proof length. That is, the proof consists of a constant number of Reed–Solomon codewords of size $O(N)$, and the running time of the prover is dominated by the time required to produce these encodings. In particular, if there were a linear-time encoding procedure for the Reed–Solomon code, our prover would run in linear time, and thereby achieve optimal prover efficiency without any other changes to the scheme itself.

Small fields. All of our results are stated over large fields. Computations over small fields (e.g. $\mathbb{F}_2$) can be handled by moving to an extension field, which introduces an additional logarithmic factor in the proof length and prover time (the same is true of [23], as well as AURORA and FRACTAL). Even with this additional logarithmic factor, our construction matches the state of the art for prover complexity (but not proof length) for succinct *boolean* circuit satisfiability, while improving verifier time to polylogarithmic.

4.1.1.1 Delegating bounded-space algebraic computation

Rank-one constraint satisfiability (R1CS) is a natural generalization of arithmetic circuits that is widely used across theoretical and applied constructions of proof systems (see [55]). An R1CS instance is specified by matrices A, B, C over a finite field $\mathbb{F}$, and is satisfied by a vector w if $Aw \circ Bw = Cw$, where $\circ$ is the element-wise (Hadamard) product. Arithmetic circuits reduce in linear time to R1CS instances.

Many problems of interest, however, involve R1CS instances where the matrices A, B, C have some structure. For example, many applications consider computations that involve checking many Merkle authentication paths — in this case a hash function is invoked many times, within the same path and across different paths. It would be valuable for the verifier to run in time that is related to a *succinct representation* of such instances, rather than to the (much larger) explicit representation that "ignores" the structure. In light of this motivation, we introduce a notion of *succinctly-represented R1CS instances* that capture a rich class of bounded-space algebraic computations. (Later in the paper we refer to these as *algebraic automata*.)

Definition 4.1.1 (informal)**.** *A* **succinct R1CS** *instance is specified by matrices $A = [A_0|A_1], B = [B_0|B_1], C = [C_0|C_1] \in \mathbb{F}^{k \times 2k}$ over $\mathbb{F}$, and time bound T, and is satisfied by a vector $z \in \mathbb{F}^{kT}$ if*

$$\begin{pmatrix} A_0 & A_1 & & & \\ & A_0 & A_1 & & \\ & & \ddots & \ddots & \\ & & & A_0 & A_1 \end{pmatrix} z \circ \begin{pmatrix} B_0 & B_1 & & & \\ & B_0 & B_1 & & \\ & & \ddots & \ddots & \\ & & & B_0 & B_1 \end{pmatrix} z = \begin{pmatrix} C_0 & C_1 & & & \\ & C_0 & C_1 & & \\ & & \ddots & \ddots & \\ & & & C_0 & C_1 \end{pmatrix} z$$

[3] In terms of bit operations, the prover runs in time $O(N' \log^2 N' \operatorname{poly}(\log \log N')) = \tilde{O}(N \log N)$, where $N' = N/\log|\mathbb{F}| = \Theta(N/\log N)$ is the size of the instance measured in field elements.

The relation Succinct-R1CS *consists of pairs* $(\mathbb{x}, z)$ *s.t.* $\mathbb{x}$ *is an instance of succinct R1CS satisfied by* z*.*

The size of an *instance* is $O(k^2 + \log T)$, but the size of the described computation is kT. Note that Succinct-R1CS is a PSPACE-complete relation, while the (regular) R1CS relation is merely NP-complete.

To obtain some intuition about the definition, consider the problem of repeated application of an arithmetic circuit $\mathcal{C}\colon \mathbb{F}^n \to \mathbb{F}^n$. Suppose that we want to check that there exists z such that $\mathcal{C}^T(z) = 0^n$, where $\mathcal{C}^T = \mathcal{C}(\mathcal{C}(\cdots\mathcal{C}(\cdot)))$ is the circuit that applies $\mathcal{C}$ iteratively T times. The circuit $\mathcal{C}^T$ has size $\Omega(|\mathcal{C}| \cdot T)$, and if the verifier were to "unroll" the circuit then it would pay this cost in time. However, the R1CS instance corresponding to $\mathcal{C}^T$ is of the above form, with $k = O(|\mathcal{C}|)$, where (roughly) the matrices A_0, B_0, C_0 represent the gates of $\mathcal{C}$ and A_1, B_1, C_1 represent the wires between adjacent copies of $\mathcal{C}$. (The condition that the output of $\mathcal{C}^T$ is zero is encoded separately as a "boundary constraint"; see Definition 4.7.1 for details.)

Our first result gives a constant-round IOP for satisfiability of succinct R1CS where the verifier runs in time $\mathrm{poly}(k, \log T)$, the prover has arithmetic complexity $O(kT \log kT)$, and the proof length is $O(kT \log |\mathbb{F}|)$ (linear in the computation transcript). In the theorem statement below we take $k = O(1)$ for simplicity.

Theorem 1 (informal)**.** *There is a universal constant* $\epsilon_0 \in (0, 1)$ *such that, for any computation time bound* $T(n)$ *and large smooth field family* $\mathbb{F}(n)$*, there is a* 4*-round IOP for succinct R1CS over* $\mathbb{F}(n)$*, with proof length* $O(T(n) \log |\mathbb{F}(n)|)$*,* 4 *queries, and soundness error* ϵ_0*. The prover uses* $O(T(n) \log T(n))$ *field operations and the verifier uses* $\mathrm{poly}(n, \log T(n))$ *field operations.*

As in prior work (e.g., [41]), "large smooth field" refers to a field of size $\Omega(N)$, whose additive or multiplicative group has a nice decomposition (see Definitions 4.9.2 and 4.9.3). For example, ensembles of large enough binary fields have this property, as well as prime fields with smooth multiplicative groups.

4.1.1.2 Delegating unbounded-space algebraic computation

While algebraic automata capture a useful class of computations, they are restricted to space-bounded computation (recall Succinct-R1CS $\in$ PSPACE). In particular, they do not capture general NEXP computations unless PSPACE $=$ NEXP. Our second result shows that a NEXP-complete algebraic problem, namely a succinct version of the *arithmetic circuit satisfiability* problem over large fields, has efficient IOPs.

Succinct SAT. In more detail, first recall the NP-complete problem of *arithmetic* circuit satisfiability, denoted ASAT, in which the goal is to decide whether an arithmetic circuit $C\colon \mathbb{F}^k \to \mathbb{F}$ over a finite field $\mathbb{F}$ has a satisfying assignment or not. Then, to capture NEXP, we follow Papadimitriou and Yannakakis [123] and consider a *succinct* version of the arithmetic circuit satisfiability problem, denoted Succinct-ASAT.[4] In this variant of the problem, we consider the

[4]Informally, Papadimitriou and Yannakakis [123] showed that if a language $\mathcal{L}$ is NP-complete under a local reduction, then its succinct version Succinct-$\mathcal{L}$ is NEXP-complete. A reduction is *local* if each bit of the reduction's output can be computed from a poly-logarithmic number of bits of the input, in poly-logarithmic time (which is typically the case for NP-completeness reductions).

satisfiability of circuits of size N that are represented by circuit descriptors of size $O(\log N)$. These descriptors are themselves circuits that, given a gate index g, return the type of gate g (addition or multiplication), its left input gate, and its right input gate.

Definition 4.1.2 (informal)**.** *The relation* Succinct-ASAT *consists of pairs* $\big((\mathbb{F}, m, H, I, o, D), w\big)$*, where* $\mathbb{F}$ *is a finite field,* $m \in \mathbb{N}$*,* H *is a succinctly-represented subset of* $\mathbb{F}^m$ *representing the gates of the circuit,* I *is a subset of* H *representing the input gates,* $o \in H$ *is the output gate,* $D\colon H \to (\{+, \times\} \times H \times H) \cup \{\mathbb{F}\}$ *is a circuit descriptor of an arithmetic circuit* C*, and the witness* $w\colon I \to \mathbb{F}$ *is such that* $C(w) = 0$*.*

Our second result is an IOP for Succinct-ASAT that has linear proof length and constant query complexity. We stress that the verifier in this IOP runs in *(poly)logarithmic* time in the size N of the circuit.[5]

Theorem 2 (informal)**.** *There is a universal constant* $\epsilon_0 \in (0, 1)$ *such that there is a* 5*-round IOP for* Succinct-ASAT *over large smooth fields with proof length* $O(N)$*,* 5 *queries, and soundness error* ϵ_0*. The prover uses* $O(N \log N)$ *field operations and the verifier uses* $\mathrm{poly}(|D|, \log N)$ *field operations.*

As in prior work (e.g., [41]), "large smooth field" refers to a field of size $\Omega(N)$, whose additive or multiplicative group has a nice decomposition (see Definition 4.9.2). For example, ensembles of large enough binary fields have this property. A *round* consists of a verifier message followed by a prover (oracle) message.

The single logarithmic factor in the prover's arithmetic complexity solely comes from the use of a constant number of Fast Fourier transforms over domains of size $\Theta(N)$. This prover time matches the best prover times of protocols for *non-succinct* arithmetic circuit satisfiability (which is merely NP-complete).

We remark that standard query reduction techniques do *not* preserve linear proof length here. Nevertheless, they can be modified in a straightforward way to preserve it, at the cost of an additional round of interaction. In particular, the number of queries in Theorem 2 can be reduced to 2 at the cost of an additional round.

Algebraic machines. We prove Theorem 2 by designing an IOP for the *algebraic machine* relation. This is a natural algebraic analogue of the bounded accepting problem for nondeterministic random-access machines, where the transition function is an arithmetic (rather than Boolean) circuit. There is a simple linear-size reduction from Succinct-ASAT to the satisfiability problem for algebraic machines: the machine holds the values of the wires in its memory, and checks that each gate is correctly evaluated.

Theorem 3 (informal)**.** *There is a universal constant* $\epsilon_0 \in (0, 1)$ *such that, for any computation time bound* $T(n)$ *and large smooth field* $\mathbb{F}(n)$*, there is a* 5*-round IOP for the satisfiability problem of* $T(n)$*-time algebraic machines over* $\mathbb{F}(n)$*, with proof length* $O(T(n) \log |\mathbb{F}(n)|)$*,* 5 *queries, and soundness error* ϵ_0*. The prover uses* $O(T(n) \log T(n))$ *field operations and the verifier uses* $\mathrm{poly}(n, \log T(n))$ *field operations.*

[5] This is because the verifier runs in time polynomial in the size of the circuit descriptor D, which in turn is logarithmic in the size of the circuit C that it describes.

For simplicity, we have stated Theorem 3 for machines whose description is a constant number of field elements, or $\Theta(\log |\mathbb{F}|)$ bits. The proof length is *linear* in the size of the computation trace, which is $N := \Theta(T \log |\mathbb{F}|)$ bits. We stress that the number of queries is 5, *regardless* of the machine.

On the power of machines. In the linear-length regime, the choice of computational model supported by a proof protocol is important, because reductions between problems typically introduce logarithmic factors. For example, it is not known how to reduce a random-access machine, or even a Turing machine, to a circuit of linear size. Indeed, the linear-size sublinear-query PCP of [38] only works for circuit but not machine computations. We thus view Theorem 3 as particularly appealing, because it achieves linear length for a powerful model of computation, algebraic machines, which facilitates linear-size reductions from many other problems. Notably, while Theorem 3 implies Theorem 2, *we do not know whether the converse holds*. We view the identification of a model which is both highly expressive and amenable to efficient probabilistic checking using IOPs as a contribution of this work.

	rounds	**type**	**prover time**	**verifier time**	**proof length**	**queries**
[119]	1	SB	$N \operatorname{polylog}(N)$ *	$\operatorname{polylog}(N)$	$N \operatorname{polylog}(N)$	$O(1)$
[38]	1	B	$\operatorname{poly}(N)$ †	$\operatorname{poly}(N)$ †	$O_\epsilon(N)$	N^ϵ
[25]	3	B	$\operatorname{poly}(N)$	$\operatorname{poly}(N)$	$O(N)$	$O(1)$
[23]	$O(\log N)$	SA ◊	$\widetilde{O}(N \log^2 N)$ ‡	$\operatorname{polylog}(N)$	$O(N \log N)$	$O(\log N)$
AURORA	$O(\log N)$	A ◊	$\widetilde{O}(N \log N)$ ‡	$\operatorname{poly}(N)$	$O(N)$	$O(\log N)$
this chapter	5	SA ◊	$\widetilde{O}(N \log N)$ ‡	$\operatorname{polylog}(N)$	$O(N)$	5

Figure 4.1: Comparison of PCP/IOP constructions for circuit satisfiability problems for a (fixed) constant soundness. Here N is the size of the circuit *in bits*, which means that, for arithmetic circuits, N implicitly includes a factor of $\log |\mathbb{F}|$. For succinct problems, the circuit size N is exponential in the size of its description. "Type" refers to the type of circuit supported by the construction: "S" denotes "succinct", "B" denotes "boolean" and "A" denotes "arithmetic".
*: [119] shows a $\operatorname{poly}(N)$ bound; this tighter bound is due to [28].
◊: The size of the underlying field must grow as $\Omega(N)$ to achieve the stated efficiency. Problems over smaller fields (e.g. boolean circuits) incur a multiplicative cost of $\log N$ in prover time and proof length.
†: The specified time is for *non-uniform* computation (each input size receives $\operatorname{poly}(N)$ advice bits).
‡: The notation $\widetilde{O}$ hides $\operatorname{poly}(\log \log N)$ factors, which arise because here we consider the *bit complexity* of the prover (rather than the arithmetic complexity).

4.1.2 Limitations of prior work

There are relatively few works that explicitly deal with prover complexity for PCP and IOP constructions. We present a comparison of the relevant parameters for each construction in Fig. 4.1. Since we are concerned with logarithmic factors, it is not sufficient to specify only a

complexity class (NP or NEXP) for each one. Instead, for each proof system we give a canonical expressive language for which the given parameters are achieved. In particular, the first three proof systems are for boolean circuit problems, and the latter three are for arithmetic circuit problems. For purposes of comparison, all of the parameters for both boolean and arithmetic constructions are presented in terms of bit complexity.

Our construction also achieves asymptotically optimal proof length and query complexity, which are more well-studied metrics. There are two natural approaches that one could take to achieve such a result: (1) start from a construction with constant query complexity and reduce proof length; or (2) start from a construction with linear proof length and reduce query complexity. We summarize prior works that have followed these approaches, and highlight the limitations that arise in each case.

Approach (1). The first approach has been studied extensively [16, 126, 102, 42, 88, 37], leading to PCPs for NEXP with proof length $N \operatorname{polylog}(N)$ and query complexity $O(1)$ [41, 36, 77, 119]. Later works have reduced the logarithmic factors in the proof length [28, 27], but attempts to achieve linear length have failed. Recent work has obtained IOPs with proof length $O(N \log N)$ but at the cost of increasing query complexity from $O(1)$ to $O(\log N)$ [21, 23].

Approach (2). The second approach has received much less attention. Insisting on linear proof length significantly restricts the available techniques because many tools introduce logarithmic factors in proof length. For example, one cannot rely on arithmetization via multivariate polynomials and standard low-degree tests, nor rely on algebraic embeddings via de Bruijn graphs for routing; in addition, query-reduction techniques for interactive PCPs [106] do not apply to the linear proof length regime. The state-of-the-art in linear-length PCPs is due to [38], and the construction is based on a non-uniform family of algebraic geometry (AG) codes (every input size needs a polynomial-size advice string). In more detail, [38] proves that for every $\epsilon \in (0, 1)$ there is a (non-uniform) PCP for the NP-complete problem CSAT (Boolean circuit satisfiability) with proof length $2^{O(1/\epsilon)}N$ and query complexity N^{ϵ}, much more than our goal of $O(1)$.

By leveraging interaction, [25] obtains IOPs for CSAT with proof length $O(N)$ and query complexity $O(1)$. This is a natural starting point for our goal of achieving polylogarithmic-time verification, because we are "only" left to extend this result from CSAT to its succinct analogue, Succinct-CSAT. Unfortunately, the construction in [25] uses AG codes and such an extension would, in particular, require obtaining a succinct representation of a dense asymptotically good family of AG codes over a small field, which is out of reach of current techniques. More generally, we do not know of any suitable code over small fields, which currently seems to prevent us from obtaining linear-size IOPs for Succinct-CSAT.

We now consider arithmetic circuit satisfiability defined over fields $\mathbb{F}$ that are large (of size $\Omega(N)$). In this regime, we have seen IOPs for ASAT with proof length $O(N)$ and query complexity $O(\log N)$, in Chapter 3. The arithmetization, following [41], is based on the Reed–Solomon code and uses the algebraic structure of large smooth fields. Testing is done via FRI [22], a recent IOP of proximity for the Reed–Solomon code with linear proof length and logarithmic query complexity. The construction in Chapter 3, which we will build upon, falls short of our goal on two fronts: verifier time is linear in the size of the circuit rather than polylogarithmic, and query complexity is logarithmic rather than constant.

Comparison with [21, 23]. The techniques that we use in this work to achieve linear proof length could be applied to the protocols of [21, 23], which would yield linear-size proofs there. In this modified protocol, however, in order to verify time-T computations the verifier must read $O(\log T)$ field elements from the proof (see e.g. [23, Lemma B.9]); in our protocol, the number of field elements the verifier reads is a universal constant. This is because the query complexity of these protocols depends linearly on the size of the description of the transition function of the machine (e.g., for Succinct-ASAT, this would be the size of the circuit descriptor).

4.1.3 Open questions

We highlight four problems left open by our work.

Optimal arithmetic complexity. The prover in our construction has strictly quasilinear arithmetic complexity and produces a proof of linear size. A natural question is whether the arithmetic complexity of the prover can be reduced to linear. To do so with our construction would require a breakthrough in encoding algorithms for the Reed–Solomon code. A promising direction may be to build IOPs based on codes with linear-time encoding procedures [139, 101, 51].

All fields. The question of whether it is possible to simultaneously achieve linear-length proofs and polylogarithmic-time verifier for Succinct-ASAT over *any* field $\mathbb{F}$ remains open. Progress on this question motivates the search for arithmetization-friendly families of good codes beyond the Reed–Solomon code. For example, the case of $\mathbb{F} = \mathbb{F}_2$, which corresponds to boolean circuits, motivates the search for succinctly-represented families of good algebraic-geometry codes over constant-size fields with fast encoding algorithms.

Zero knowledge. Zero knowledge, while not a goal of this work, is a desirable property, because zero knowledge PCP/IOPs lead to zero knowledge succinct arguments [104, 32]. Straightforward modifications to the protocol, similar to those described in Section 3.2.3, achieve a notion of zero knowledge wherein the simulator runs in time polynomial in the size of the computation being checked, which is meaningful for nondeterministic problems since it does not have access to the witness.

There is a stronger notion of zero knowledge for succinct languages where the simulator runs in *polylogarithmic* time (in time polynomial in the size of the instance). This gap was precisely the subject of a work on designing succinct simulators for certain tests [24]. Whether low-degree tests with the parameters we require have succinct simulators remains an intriguing problem that we leave to future research.

Round complexity. Our protocol has 5 rounds. Round complexity can be reduced to 4 at the cost of increased (constant) query complexity. Reducing round complexity beyond this while preserving linear proof length and polylogarithmic time verification, or finding evidence against this possibility, remains open.

4.2 Technical overview

We discuss the main ideas behind our results. Recall that Succinct-ASAT is the satisfiability problem for succinctly-represented arithmetic circuits over the field $\mathbb{F}$; this problem

is NEXP-complete for any field $\mathbb{F}$ [123]. Our goal is to construct an IOP for Succinct-ASAT, over a large field $\mathbb{F}$, with proof length that is linear in the size of the circuit N and strictly quasilinear ($O(N \log N)$) prover arithmetic complexity; crucially, the running time of the verifier is *polylogarithmic* in the size of the circuit (more precisely, polynomial in the size of the circuit descriptor). Additionally, we strive to optimize the query and round complexity of this IOP. We stress that no prior work achieves non-trivial linear-length PCPs or IOPs wherein the verifier runs in polylogarithmic time in the size of the circuit.

The rest of this section is organized as follows. In Section 4.2.1 we discuss where the construction in Chapter 3 fails to achieve polylogarithmic verification. In Section 4.2.2 we discuss our approach to overcoming the limitations of prior work by describing a new protocol for checking succinctly-represented linear relations; this achieves an *exponential* improvement over the prior state of the art. In Section 4.2.4 we discuss how to overcome the challenges that arise when attempting to build on this exponential improvement to checking the computation of algebraic automata, and then of algebraic machines (which capture Succinct-ASAT as a special case). In Section 4.2.5 we describe a modular framework, which we call *oracle reductions*, in which we prove our results.

4.2.1 Our starting point

The IOP designed in this chapter is built upon the AURORA IOP from Chapter 3. Recall that the AURORA IOP is obtained by combining the *lincheck protocol* (Section 3.5) and the *rowcheck protocol* (Section 2.3.4).

While on the one hand the verifier in the rowcheck protocol runs in time that is polylogarithmic in $|H|$ (which is good) the verifier in the lincheck protocol runs in time that is linear in $|H|$ (which is much too slow). In other words, if we simply invoked the AURORA IOP on the circuit described by an instance of Succinct-ASAT, the verifier would run in time that is linear in the circuit size, which is exponentially worse than our goal of polylogarithmic time. This state of affairs is the starting point of this chapter.

Next, in Section 4.2.2, we discuss how to obtain a succinct lincheck protocol that, for suitable linear relations, is exponentially more efficient. After that, in Section 4.2.4, we discuss how to build on our succinct lincheck protocol to reduce Succinct-ASAT to testing membership in the Reed–Solomon code, while achieving verifier time that is *polylogarithmic* in circuit size.

Throughout, we present our contributions as *oracle reductions* from some computational task to testing membership in the Reed–Solomon code. Loosely speaking, these are reductions in the setting of the IOP model (and therefore, in particular, allow interaction in which the prover sends PCP oracles). This abstraction allows us to decouple IOP protocol-design from the low-degree test that we invoke at the end of the protocol. They generalise the RS-encoded IOP framework described in Section 5.4.1. See Section 4.2.5 for details.

4.2.2 Checking succinctly-represented linear relations

Following the above discussion, we now temporarily restrict our attention to devising a lincheck protocol, which reduces checking linear relations defined by matrices $M \in \mathbb{F}^{H \times H}$ to testing membership in the Reed–Solomon code, in which the verifier runs in time that is

polylogarithmic in $|H|$. This is not possible in general, however, because the verifier needs to at least *read the description* of the matrix M. We shall have to consider matrices M that have a special structure that can be exploited to obtain an exponential improvement in verifier time. This improvement is a core technical contribution of this paper, and we refer to the resulting reduction as the *succinct lincheck protocol*. We start by briefly recalling the lincheck protocol from Section 3.5.

Definition 4.2.1 (informal)**.** *In the lincheck problem, we are given a subset $H \subseteq \mathbb{F}$, Reed–Solomon codewords $f, g \in \mathrm{RS}[L, \rho]$ encoding vectors $x, y \in \mathbb{F}^H$, and a matrix $M \in \mathbb{F}^{H \times H}$. The goal is to check that $x = My$.*

A simple probabilistic test for the claim "$x = My$" is to check that $\langle r, x - My \rangle = 0$ for a random $r \in \mathbb{F}^H$. Indeed, if $x \neq My$, then $\Pr_{r \in \mathbb{F}^H}[\langle r, x - My \rangle = 0] = 1/|\mathbb{F}|$. However, this approach would require the verifier to sample, and send to the prover, $|H|$ random field elements (too many).

A natural derandomization is to choose $\vec{r}$ using a small-bias generator over $\mathbb{F}$, rather than uniformly at random. A small-bias generator G over $\mathbb{F}$ is a function with the property that for any nonzero $z \in \mathbb{F}^H$, it holds with high probability over $\rho \in \{0,1\}^\ell$ that $\langle z, G(\rho) \rangle \neq 0$. Now the verifier needs to send only ℓ bits to the prover, which can be much smaller than $|H| \log |\mathbb{F}|$.

A natural choice (used also, e.g., in [16, §5.2]) is the powering construction of [6], which requires sending a *single* random field element ($\ell = \log |\mathbb{F}|$), and incurs only a modest increase in soundness error. In this construction, we define a vector $\vec{r}(X) \in \mathbb{F}[X]^H$ of linearly independent polynomials in X, given by $(1, X, X^2, \ldots, X^{|H|-1})$. The small-bias generator is then $G(\alpha) := \vec{r}(\alpha)$ for $\alpha \in \mathbb{F}$. If z is nonzero then $h(X) := \langle \vec{r}(X), z \rangle$ is a nonzero polynomial and so $\Pr_{\alpha \in \mathbb{F}}[\langle G(\alpha), z \rangle = 0] \leq \deg(h)/|\mathbb{F}|$. The verifier now merely has to sample and send $\alpha \in \mathbb{F}$, and the prover must then prove the claim "$h(\alpha) = 0$" to the verifier. Rearranging, this is the same as testing that $\langle \vec{r}(\alpha), x \rangle - \langle \vec{r}(\alpha) M, y \rangle = 0$. The problem is thus reduced to checking inner products of known vectors with oracles.

In the setting of Reed–Solomon codewords, if f_u is an encoding of u and f_v is an encoding of v, then $f_u \cdot f_v$ is an encoding of $u \circ v$, the pointwise product of u and v. Hence, to check that $\langle u, v \rangle = c$, it suffices to check that the low-degree polynomial $f_u \cdot f_v$ sums to c on H, since $\langle u, v \rangle = \sum_{h \in H} f_u(h) f_v(h)$. This can be achieved by running the *univariate sumcheck protocol* (Section 3.4) on the codeword $f_u \cdot f_v$. This protocol requires the verifier to efficiently determine the value of $f_u \cdot f_v$ at a given point in L.

The inefficiency. The foregoing discussion tells us that, to solve the lincheck problem, the verifier must determine the value of the Reed–Solomon encodings of $\vec{r}(\alpha) \circ x$ and $\vec{r}(\alpha) M \circ y$ at a given point in L. The encodings of the vectors x and y are provided (as f and g). Hence it suffices for the verifier to evaluate *low-degree extensions* of $\vec{r}(\alpha)$ and $\vec{r}(\alpha) M$ at a given point, and then perform a field multiplication.

This last step is the computational bottleneck of the protocol. In Section 3.5, the verifier evaluates the low-degree extensions of $\vec{r}(\alpha)$ and $M^\top \vec{r}(\alpha)$ via Lagrange interpolation, which requires time $\Omega(|H|)$. To make our verifier efficient, we must evaluate both low-degree extensions in time $\mathrm{polylog}(|H|)$. In particular, this requires that M be succinctly represented, since computing the low-degree extension of $\vec{r}(\alpha) M$ in general requires time linear in the number of nonzero entries in M, which is at least $|H|$.

The original lincheck protocol chooses the linearly independent polynomials $\vec{r}(X)$ to be the *standard* (or *coefficient*) basis $(1, X, \ldots, X^{|H|-1})$. For this basis, however, we do not know how to efficiently evaluate the low-degree extension of $\vec{r}(\alpha)$. This problem must be addressed *regardless* of the matrix M.

A new basis and succinct matrices. We leverage certain algebraic properties to overcome the above problem. There is another natural choice of basis for polynomials, the *Lagrange basis* $(L_{H,h}(X))_{h\in H}$, where $L_{H,h}$ is the unique polynomial of degree less than $|H|$ with $L_{H,h}(h) = 1$ and $L_{H,\alpha}(\gamma) = 0$ for all $\gamma \in H \setminus \{h\}$. We observe that the low-degree extension of $\vec{r}(\alpha) = (L_{H,h}(\alpha))_{h\in H} \in \mathbb{F}^H$ has a simple form that allows one to evaluate it in time $\mathrm{polylog}(|H|)$ provided that H is an additive or multiplicative subgroup of $\mathbb{F}$. In other words, the Lagrange basis yields a small-bias generator over $\mathbb{F}$ whose low-degree extension can be computed efficiently.

It remains to find a useful class of succinctly-represented matrices M for which one can efficiently evaluate a low-degree extension of $\vec{r}(\alpha)M \in \mathbb{F}^H$. The foregoing discussion suggests a natural condition: *if* we can efficiently compute a low-degree extension of a vector $v \in \mathbb{F}^H$ *then* we should also be able to efficiently compute a low-degree extension of the vector vM. If this holds for all vectors v, we say that the matrix M is (algebraically) *succinct*. For example, the identity matrix satisfies this definition (trivially), and so does the matrix with 1s on the superdiagonal for appropriate choices of $\mathbb{F}$ and H (see Section 4.7.1).

In sum, if we choose the Lagrange basis in the lincheck protocol and the linear relation is specified by a succinct matrix, then, with some work, we obtain a lincheck protocol where the verifier runs in time $\mathrm{polylog}(|H|)$. To check satisfiability of succinctly-represented arithmetic circuits, however, we need to handle a more general class of matrices, described next.

Succinct lincheck for semisuccinct matrices. We will relax the condition on a matrix M in a way that captures the matrices that arise when checking succinctly-described arithmetic circuits, while still allowing us to obtain a lincheck protocol in which the verifier runs in time $\mathrm{polylog}(|H|)$.

We show that the matrices that we consider are *semisuccinct*, namely, they can be decomposed into a "large" part that is succinct and a "small" part that has no special structure.[6] This structure should appear familiar, because it is analogous to how a succinctly-described circuit consists of a small arbitrary component (the circuit descriptor) that is repeatedly used in a structured way to define the large circuit. Another analogy is how in an automaton or machine computation a small, and arbitrary, transition function is repeatedly applied across a large computation.

Specifically, by "decompose" we mean that the matrix $M \in \mathbb{F}^{H\times H}$ can be written as the *Kronecker product* of a succinct matrix $A \in \mathbb{F}^{H_1\times H_1}$ and a small matrix $B \in \mathbb{F}^{H_2\times H_2}$; we write $M = A \otimes B$. (Succinctly representing a large operator like M via the tensor product of simpler operators should be a natural idea to readers familiar with quantum information.) In order for the product to be well-defined, we must supply a bijection $\Phi\colon H \to H_1 \times H_2$. If this bijection satisfies certain algebraic properties, which preserve the succinctness of the matrix A, we call it a *bivariate embedding*.

We obtain a succinct lincheck protocol for semisuccinct matrices.

[6] We actually need to handle matrices that are the *sum* of semisuccinct matrices, but we ignore this in this high-level discussion.

Lemma 4.2.2 (informal)**.** *There is a linear-length oracle reduction from the lincheck problem for semisuccinct matrices to testing membership in the Reed–Solomon code, where the verifier runs in polylogarithmic time.*

Next, we discuss how to obtain a reduction from algebraic automata (succinct R1CS) to testing membership in the Reed–Solomon code, where the verifier runs in time that is *polylogarithmic* in the circuit size, by building on our succinct lincheck protocol for semisuccinct matrices.

4.2.3 Checking bounded-space computations in polylogarithmic time

An instance of the *algebraic automaton* relation is specified by three $k \times 2k$ matrices (A, B, C) over $\mathbb{F}$, and a time bound T. A witness $f\colon [T] \to \mathbb{F}^k$ is valid if

$$\forall\, t \in [T-1] \quad Af(t, t+1) \circ Bf(t, t+1) = Cf(t, t+1) \ , \tag{4.1}$$

where $f(t, t+1) := f(t) \| f(t+1)$ is the concatenation of the consecutive states $f(t) \in \mathbb{F}^k$ and $f(t+1) \in \mathbb{F}^k$.

We use the term "algebraic automata" since one can think of A, B, C as specifying the transition relation of a computational device with k algebraic registers, and f as an *execution trace* specifying an accepting computation of the device. The relation is PSPACE-complete: it is in NPSPACE because it can be checked by a polynomial-space Turing machine with one-directional access to an exponential-size witness, and recall that NPSPACE = PSPACE; also, it is PSPACE-hard because given an arithmetic circuit specifying the transition relation of a polynomial-space machine, we can find an equisatisfiable R1CS instance in linear time.

If we view the execution trace f as a vector $f = f(1) \| \cdots \| f(T) \in \mathbb{F}^{Tk}$, then we can rewrite the condition in Eq. (4.1) via the following (possibly exponentially large) R1CS equation:

$$\begin{pmatrix} A_0 & A_1 & & & \\ & A_0 & A_1 & & \\ & & \ddots & \ddots & \\ & & & A_0 & A_1 \end{pmatrix} f \circ \begin{pmatrix} B_0 & B_1 & & & \\ & B_0 & B_1 & & \\ & & \ddots & \ddots & \\ & & & B_0 & B_1 \end{pmatrix} f = \begin{pmatrix} C_0 & C_1 & & & \\ & C_0 & C_1 & & \\ & & \ddots & \ddots & \\ & & & C_0 & C_1 \end{pmatrix} f$$

where $A_0, A_1 \in \mathbb{F}^{k \times k}$ are the first half and second half of A respectively; and likewise for B and C. We thus see that algebraic automata are equivalent to Succinct-R1CS.

The matrices in the above R1CS equation have a rigid block structure that we refer to as a *staircase*. Given the discussions in Sections 4.2.1 and 4.2.2, in order to achieve polylogarithmic verifier time, *it suffices to show that staircase matrices are semisuccinct* (or, at least, the sum of few semisuccinct matrices).

So let $S(M_0, M_1)$ be the staircase matrix of two given $k \times k$ matrices M_0, M_1 over $\mathbb{F}$. Namely, $S(M_0, M_1)$ is the $Tk \times Tk$ matrix with M_0-blocks on the diagonal and M_1-blocks on the superdiagonal. Observe that:

1. we can write the matrix with M_0-blocks on the diagonal as $I \otimes M_0$, where I is the $T \times T$ identity matrix;

2. we can write the matrix with M_1-blocks on the superdiagonal as $I^{\rightarrow} \otimes M_1$, where $I^{\rightarrow}$ is the $T \times T$ matrix with 1s on the superdiagonal.

Under an appropriate mapping from $[Tk]$ into a subset of $\mathbb{F}$, we prove that both of these matrices are semisuccinct. This tells us that $S(M_0, M_1)$ is the sum of two semisuccinct matrices:

$$S(M_0, M_1) := \begin{pmatrix} M_0 & M_1 & & & \\ & M_0 & M_1 & & \\ & & \ddots & \ddots & \\ & & & M_0 & M_1 \\ & & & & M_0 \end{pmatrix} = I \otimes M_0 + I^{\rightarrow} \otimes M_1 \in \mathbb{F}^{Tk} \times \mathbb{F}^{Tk} \ .$$

(Note that the above is not exactly the matrix structure we want, because of the extra M_0 block; we handle this technicality separately.) We obtain the following lemma.

Lemma 4.2.3 (informal)**.** *There is a linear-length oracle reduction from the algebraic automaton relation to testing membership in the Reed–Solomon code, where the verifier runs in time* $\mathrm{poly}(k, \log T)$.

4.2.4 Checking succinct satisfiability in polylogarithmic time

We outline our construction of a linear-length IOP for succinct circuit satisfiability (Succinct-ASAT) over a large smooth field $\mathbb{F}$. Informally, our strategy is as follows. First, we note that there is a simple linear-size reduction (see Section 4.9.2) from Succinct-ASAT to satisfiability of an algebraic automaton augmented with a permutation constraint; we refer to the latter as an *algebraic machine*. Then, we prove that checking such an instance can be reduced to checking an algebraic automaton *without* the permutation constraint, which can be achieved as described in the previous section. In the remainder of this section we discuss the main technical challenges that arise in this approach.

An instance of the *algebraic (R1CS) machine* relation is specified by two algebraic (R1CS) automata $(\mathcal{A}, \mathcal{A}')$. A witness (f, π), where $f \colon [T] \to \mathbb{F}^k$ is an execution trace and $\pi \colon [T] \to [T]$ is a permutation, is valid if: (1) f is a valid witness for the automaton $\mathcal{A}$, and (2) $f \circ \pi$ is a valid witness for the automaton $\mathcal{A}'$. The algebraic machine relation is NEXP-complete, as Succinct-ASAT reduces to it in linear time (see Section 4.9.2).

Execution traces for machines. Before we discuss how we reduce from the algebraic machine relation, we briefly explain why the above relation is a natural problem to consider, and in particular why it has anything to do with (random-access) machines. Recall that a random-access machine is specified by a list of instructions, each of which is an arithmetic operation, a control-flow operation, or a read/write to memory. One way to represent the execution trace for the machine is to record the state of the entire memory at each time step; for a time-T space-S computation, such an execution trace has size $\Theta(TS)$ (much more than linear!). Yet, the machine can access only a *single* memory location at each time step. Thus, instead of writing down the state of the entire memory at each time step, we could hope to only write the state of the accessed address — this would reduce the size of the trace to $\Theta(T \log S)$. The problem then is that it is no longer possible to check consistency of memory via local constraints because the same address can be accessed at any time.

Classical techniques of Gurevich and Shelah [100] tell us that one can efficiently represent an execution trace for a machine via *two* execution traces that are *permutations of one another*. Informally, sorting the execution trace by time enables us to check the transition relation of the machine; and sorting the execution trace by the accessed addressed (and then by time) enables us to locally check that memory is consistent. (One must ensure that, for each location, if we write some value to memory and then read the same address, we retrieve that same value.) The transition relation and memory consistency can each be expressed individually as automata. This view of machines immediately gives rise to the algebraic machine relation above.

Checking the algebraic machine relation. We have discussed how to check automata in Section 4.2.3, so it remains to check that the traces are permutations of one another. Historically this has been achieved in the PCP literature using algebraic embeddings of routing networks; e.g., see [27]. The problem is that this increases the size of the representation of the execution trace by at least a logarithmic factor. We instead use an interactive permutation test from the program checking literature [114, 47]. The test is based on the observation that $u \in \mathbb{F}^T$ is a permutation of $v \in \mathbb{F}^T$ if and only if the multi-sets given by their elements are equal, which is true if and only if the polynomials $p_u(X) = \prod_{i=1}^{T}(X - u_i)$ and $p_v(X) = \prod_{i=1}^{T}(X - v_i)$ are equal. Thus it suffices to evaluate each polynomial at a random point and check equality.

These polynomials require time $\Theta(T)$ to evaluate, which in our setting is exponential. Therefore the prover must assist the verifier with the evaluation. We show that evaluating this polynomial can be expressed as an algebraic automaton, and can therefore be checked again using the protocol from Section 4.2.3.

The reader may believe that by now we have reduced checking an algebraic machine to checking three instances of algebraic automata. Recall, however, that the algebraic automaton relation is PSPACE-complete, whereas the algebraic machine relation is NEXP-complete. What happened? The answer lies in the randomness used in the permutation automaton. In order to check that u is a permutation of v, the prover must first commit to u and v before the verifier chooses his evaluation point α, and then the prover sends the trace of the automaton that evaluates $p_u(\alpha), p_v(\alpha)$. This trace depends on the choice of α, and so we use interaction. This is captured by the *interactive automaton relation* (Definition 4.7.1), which is NEXP-complete; it can be checked in essentially the same way as the automaton relation described in Section 4.2.3.

We hence obtain the following lemma.

Lemma 4.2.4 (informal)**.** *There is a linear-length oracle reduction from the algebraic machine relation (hence,* Succinct-ASAT*) to testing membership in the Reed–Solomon code, where the verifier runs in time* $\mathrm{poly}(k, \log T)$.

4.2.5 Oracle reductions

Many results in this paper describe IOPs that reduce a computational problem to membership in (a subcode of) the Reed–Solomon code. We find it useful to capture this class of reductions via a precise definition. This lets us prove general lemmas about such reductions, and obtain our protocols in a modular fashion.

We thus formulate a new notion that we call *interactive oracle reductions* (in short, *oracle reductions*). Informally, an oracle reduction is a protocol that reduces from a computational

problem to testing membership in a code (in this paper, the code is the interleaved Reed–Solomon code). This is a well-understood idea in constructions of PCPs and IOPs. Our contribution is to provide a formal framework for this technique.

We illustrate the notion of oracle reductions via an example. Consider the problem of testing proximity to the *vanishing Reed–Solomon code*, which plays an important role in a PCP of Ben-Sasson and Sudan [41] and several other PCPs/IOPs. Informally, the goal is to test whether a univariate polynomial f, provided as an oracle, is zero everywhere on a subset H of $\mathbb{F}$.

We describe an oracle reduction that maps the foregoing problem to the problem of testing membership in the Reed–Solomon code of the related polynomial $g := f/\mathbb{Z}_H$. Observe that f is divisible by $\mathbb{Z}_H$ if and only if f is zero everywhere in H, and so g is in the Reed–Solomon code if and only if f satisfies the desired property. But what exactly *is* g? In the oracle reduction framework, we refer to g as a *virtual oracle*: an oracle whose value at any given point in its domain can be determined efficiently by making a small number of queries to concrete oracles. In this case, so long as the domain L we choose for g does not intersect H, a verifier can evaluate g at any point $\alpha \in L$ with only a single query to f. To test that g is low degree, the verifier can invoke any low-degree test on g, and simulate queries to the virtual oracle g via queries to f.

The two main parameters in an oracle reduction are the *proof length*, which is simply the total length of the oracles sent by the prover, and the *locality*, which is the number of queries one would have to make to the concrete oracles to answer a single query to any virtual oracle (in this paper, locality always equals the number of rounds). Using the perspective of oracle reductions, our main theorems (Theorems 1 and 2) follows by combining two main sub-components: (1) a linear-length 3-local oracle reduction from the Succinct-ASAT problem to proximity testing to the Reed–Solomon code (discussed in Section 4.2.4); and (2) a linear-length 3-query IOP for testing proximity to the Reed–Solomon code from [25] (see Lemma 4.9.1).

4.3 Roadmap

Fig. 4.2 below provides a diagram of the results proved in this chapter. The remaining sections in this chapter are organized as follows. In Section 4.4 we define oracle reductions, and prove how to create IOP protocols from RS oracle reductions and RS proximity tests. In Section 4.5 we define and construct trace embeddings. In Section 4.6 we describe our succinct lincheck protocol. In Section 4.7 we describe an oracle reduction from R1CS automata to testing proximity to the Reed–Solomon code, proving Theorem 1. In Section 4.8 we describe an oracle reduction from R1CS machines to testing proximity to the Reed–Solomon code. In Section 4.9 we prove Theorem 2 and Theorem 3.

4.4 Oracle reductions

We define *interactive oracle reductions* (henceforth just *oracle reductions*), which, informally, are reductions from computational problems to the problem of testing membership of collections of oracles in a code.

The main result in this section is Lemma 4.4.4 (and an implication of it, Corollary 4.4.9),

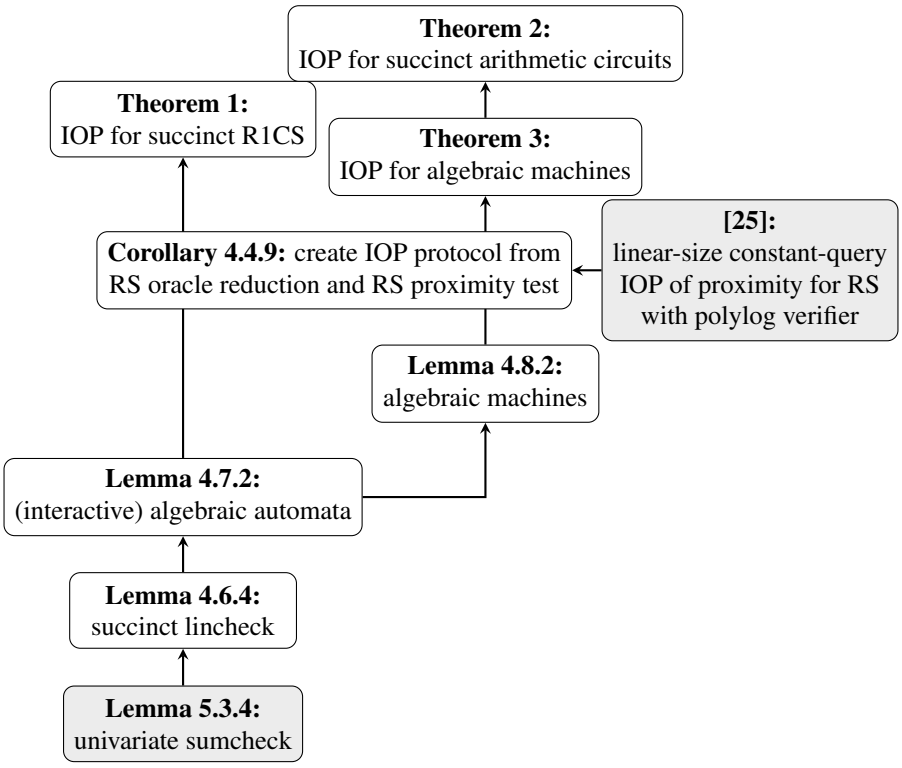


Figure 4.2: Diagram of the results in this chapter.

which enables the construction of IOPs by modularly combining oracle reductions and proximity tests. The ideas underlying oracle reductions are not new. Essentially all known constructions of PCPs/IPCPs/IOPs consist of two parts: (1) an encoding, typically via an algebraic code, that endows the witness with robust structure (often known as *arithmetization*); and (2) a procedure that locally tests this encoding (often known as *low-degree testing*).

Oracle reductions provide a formal method of constructing proof systems according to this framework. We use them to express results in Sections 4.6 to 4.8, which significantly simplifies exposition. Additionally, expressing our results as oracle reductions enables us to consider the efficiency of the oracle reduction itself as a separate goal from the efficiency of the low-degree test. In particular, future improvements in low-degree testing will lead immediately to improvements in our protocols.

This section has two parts: in Section 4.4.1 we define oracle reductions; then in Section 4.4.2, we introduce a special case of oracle reductions where the target code is the Reed–Solomon (RS) code. For this special case we give additional lemmas: we show that it suffices to prove a weaker soundness property, because it generically implies standard soundness; also, we show that all such oracle reductions admit a useful optimization which reduces the number of low-degree tests needed to a single one.

4.4.1 Definitions

Informally, an oracle reduction is an interactive public-coin protocol between a prover and a verifier that reduces membership in a language to a promise problem on oracles sent by the prover during the protocol.

In more detail, an oracle reduction from a language $\mathcal{L} \subseteq X$ to a relation $\mathcal{R}' \subseteq X' \times \Sigma^s$ is an interactive protocol between a prover and a verifier that both receive an instance $\mathbb{x} \in X$, where in each round the verifier sends a message and the prover replies with an oracle (or several oracles), as in the IOP model. Unlike in an IOP, the verifier *does not make any queries*. Instead, after the interaction the verifier outputs a list of claims of the form "$(\mathbb{x}, \mathbf{\Pi}) \in \mathcal{R}'$", which may depend on the verifier's randomness, where $\mathbb{x}' \in X'$ and $\mathbf{\Pi}$ is a deterministic oracle algorithm that specifies a string in Σ^s as follows: the i-th entry in Σ^s is computed as $\mathbf{\Pi}^{\pi_1,\ldots,\pi_r}(i)$, where π_j is the oracle sent by the prover in the j-th round. The reduction has the property that if $\mathbb{x} \in \mathcal{L}$ then all claims output by the verifier are true, and if instead $\mathbb{x} \notin \mathcal{L}$ then (with high probability over the verifier's randomness) at least one claim is false.

We refer to each oracle algorithm $\mathbf{\Pi}^{(j)}$ as a *virtual oracle* because $\mathbf{\Pi}^{(j)}$ represents an oracle that is derived from oracles sent by the prover. We are interested in virtual oracles $\mathbf{\Pi}^{(j)}$ where, for each i, the number of queries $\mathbf{\Pi}^{\pi_1,\ldots,\pi_r}(i)$ makes to the oracles is small. For simplicity, we also assume that the algorithms are non-adaptive in that the queried locations are independent of the answers to the queries.

A crucial property is that virtual oracles with small locality compose well, which allows us to compose oracle reductions. For this we need an oracle reduction *of proximity* (Definition 4.4.3), which we can view as an oracle reduction from a relation $\mathcal{R} \subseteq X \times \Sigma^s$ to another relation $\mathcal{R}' \subseteq X' \times \Sigma^{s'}$. Then we can construct an oracle reduction from $\mathcal{L}$ to $\mathcal{R}'$ by composing an oracle reduction A from $\mathcal{L}$ to $\mathcal{R}'$ with an oracle reduction of proximity B from $\mathcal{R}'$ to $\mathcal{R}''$. Such a reduction may output virtual oracles of the form $\mathbf{\Pi}_B^{\mathbf{\Pi}_A}$ where $\mathbf{\Pi}_B$ is a virtual oracle output by B and $\mathbf{\Pi}_A$ is a virtual oracle output by A. This can be expressed as a standard virtual oracle with access to the prover messages, and if $\mathbf{\Pi}_A$ and $\mathbf{\Pi}_B$ have small locality then so does $\mathbf{\Pi}_B^{\mathbf{\Pi}_A}$.

We now formalize the foregoing discussion, starting with the notion of a virtual oracle. Since the virtual oracles in this work are non-adaptive, we specify them via query ("pre-processing") and answer ("post-processing") algorithms. The query algorithm receives an index $i \in [s]$ and computes a list of locations to be queried across oracles. The answer algorithm receives the same index i, and answers to the queries, and computes the value of the virtual oracle at location i. In other words, the answer algorithm computes the value of the virtual oracle at the desired location from the values of the real oracles at the queried locations.

Definition 4.4.1. *A* **virtual oracle** $\mathbf{\Pi}$ *of length* s *over an alphabet* Σ *is a pair of deterministic polynomial-time algorithms* (Q, A)*. Given any oracles* $\pi_1, \ldots, \pi_r$ *of appropriate sizes, these algorithms define an oracle* $\mathbf{\Pi} \in \Sigma^s$ *given by* $\mathbf{\Pi}[\pi_1, \ldots, \pi_r](i) := A(i, (\pi_j[k])_{(j,k)\in Q(i)})$ *for* $i \in [s]$*.* $\mathbf{\Pi}$ *is* ℓ**-local** *if* $\max_{i\in[s]} |Q(i)| \leq \ell$*.*

Observe that the definition of a virtual oracle given above is equivalent to saying that $\mathbf{\Pi}$ is an algorithm with non-adaptive query access to $\pi_1, \ldots, \pi_r$. Where convenient we will use this perspective.

We now define the notion of an oracle reduction. Since in this work we primarily deal with relations, rather than languages, we define our reductions accordingly.

Definition 4.4.2. *An* **oracle reduction** *from a relation $\mathcal{R}$ to a relation $\mathcal{R}'$ with base alphabet Σ is an interactive protocol between a prover P and verifier V that works as follows. The prover P takes as input an instance-witness pair $(\mathbb{x}, \mathbb{w})$ and the verifier V takes as input the instance $\mathbb{x}$. In each round, V sends a message $m_i \in \{0,1\}^*$, and P replies with an oracle $\pi_i \in \Sigma_i^*$ over an alphabet $\Sigma_i = \Sigma^{s_i}$; let $\pi_1, \ldots, \pi_r$ be all oracles sent.[7] After the interaction, V outputs a list of instances $(\mathbb{x}^{(1)}, \ldots, \mathbb{x}^{(m)})$ and a list of virtual oracles $(\mathbf{\Pi}^{(1)}, \ldots, \mathbf{\Pi}^{(m)})$ over alphabets $\Sigma'_1, \ldots, \Sigma'_m$ respectively, where $\Sigma'_i = \Sigma^{s'_i}$.*

We say that the oracle reduction has **soundness error** ϵ *and* **distance** δ *if the following conditions hold.*

- Completeness: *If $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}$ then, with probability 1 over the verifier's randomness, for every $j \in [m]$ it holds that $\left(\mathbb{x}^{(j)}, \mathbf{\Pi}^{(j)}[\pi_1, \ldots, \pi_r]\right) \in \mathcal{R}'$ where $(\mathbb{x}^{(j)}, \mathbf{\Pi}^{(j)})_{j\in[m]} \leftarrow (P(\mathbb{x}, \mathbb{w}), V(\mathbb{x}))$.*

- Soundness:[8] *If $\mathbb{x} \notin \mathcal{L}(\mathcal{R})$ then for any prover $\tilde{P}$, with probability $1 - \epsilon$ over the verifier's randomness, there exists $j \in [m]$ such that $\Delta(\mathbf{\Pi}^{(j)}[\pi_1, \ldots, \pi_r], \mathcal{R}'|_{\mathbb{x}^{(j)}}) > \delta$ where $(\mathbb{x}^{(j)}, \mathbf{\Pi}^{(j)})_{j\in[m]} \leftarrow (P(\mathbb{x}, \mathbb{w}), V(\mathbb{x}))$.*

An oracle reduction is *public coin* if all of the verifier's messages consist of uniform randomness. All of the oracle reductions we present in this paper are public coin. Note that we can always choose the base alphabet Σ to be $\{0,1\}$, but it will be convenient for us to use a larger base alphabet.

This above definition can be viewed as extending the notion of *linear-algebraic CSPs* [26], and indeed Lemma 4.4.4 below gives a construction similar to the "canonical" PCP described in that work.

It will be useful to *compose* oracle reductions. As in the PCP setting, for this we will require an object with a stronger *proximity soundness* property.

Definition 4.4.3. *An* **oracle reduction of proximity** *is as in Definition 4.4.2 except that, for a given proximity parameter $\delta_0 \in (0,1)$, the soundness condition is replaced by the following one.*

- Proximity soundness: *If $(\mathbb{x}, \mathbb{w})$ is such that $\Delta(\mathbb{w}, \mathcal{R}|_{\mathbb{x}}) > \delta_0$ then for any prover $\tilde{P}$, with probability $1-\epsilon$ over the verifier's randomness, $\exists\, j \in [m]$ such that $\Delta(\mathbf{\Pi}^{(j)}[\pi_1, \ldots, \pi_r], \mathcal{R}'|_{\mathbb{x}^{(j)}}) > \delta(\delta_0)$ where $(\mathbb{x}^{(j)}, \mathbf{\Pi}^{(j)})_{j\in[m]} \leftarrow (P(\mathbb{x}, \mathbb{w}), V(\mathbb{x}))$.*

In the PCPP literature the foregoing soundness property is usually known as *robust soundness*, and the condition is expressed in terms of expected distance. The definition given here is more convenient for us.

Efficiency measures. There are several efficiency measures that we study for an oracle reduction.

[7] Sometimes it is convenient to allow the prover to reply with multiple oracles $\pi_{i,1}, \pi_{i,2}, \ldots$; all discussions extend to this case.

[8] This is analogous to the "interactive soundness error" ε_{i} in Section 2.3.1.

- An oracle reduction has r *rounds* if the interactive protocol realizing it has r rounds.
- An oracle reduction has m *virtual oracles* and *locality* ℓ if the verifier outputs at most m virtual oracles $\{\mathbf{\Pi}^{(j)} = (Q_j, A_j)\}_{j\in[m]}$, and it holds that $\max_{i\in[s]} |\cup_{j=1}^{m} Q_j(i)| \leq \ell$. Note that the answer to a single query may consist of multiple symbols over the base alphabet Σ, but we count the query only once.
- An oracle reduction has *length* $s = \sum_{i=1}^{r} s_i|\pi_i|$ over the base alphabet. Its length in *bits* is $s \log |\Sigma|$.

Other efficiency measures include the running time of the prover and of the verifier.

Oracle reductions combine naturally with proofs of proximity to produce IOPs. The following lemma is straightforward, and we state it without proof.

Lemma 4.4.4. *Suppose that there exist:*
(i) an r-round oracle reduction from $\mathcal{R}$ to $\mathcal{R}'$ over base alphabet Σ with soundness error ϵ, distance δ, length s, locality ℓ, and m virtual oracles;
(ii) an r'-round IOPP for $\mathcal{R}'$ over alphabet Σ with soundness error ϵ', proximity parameter $\delta' \leq \delta$, length s', and query complexity $(\mathsf{q}_\mathsf{w}, \mathsf{q}_\pi)$.
Then there exists an $(r + mr')$-round IOP for $\mathcal{R}$ with soundness error $\epsilon + \epsilon'$, length $s + s' \cdot m$ over Σ, and query complexity $(\mathsf{q}_\mathsf{w} \cdot \ell + \mathsf{q}_\pi) \cdot m$.

4.4.2 Reed–Solomon oracle reductions

In this work we focus on a special class of oracle reductions, in which we reduce to membership in the Reed–Solomon code, and where the virtual oracles have a special form. These reductions coincide with "RS-encoded IOPs" as defined in Section 2.3.3, which we recast in the language of virtual oracles.

We first define the notion of a *rational constraint*, a special type of virtual oracle that is "compatible" with the (interleaved) Reed–Solomon code.

Definition 4.4.5. *A* **rational constraint** *is a virtual oracle $\mathbf{\Pi} = (Q, A)$ over a finite field $\mathbb{F}$ where $Q(\alpha) = ((1, \alpha), \ldots, (r, \alpha))$ and $A(\alpha, \beta_1, \ldots, \beta_r) = N(\alpha, \beta_1, \ldots, \beta_r)/D(\alpha)$, for two arithmetic circuits (without division gates) $N \colon \mathbb{F}^{\sum_i s_i} \to \mathbb{F}$ and $D \colon \mathbb{F} \to \mathbb{F}$.*

A Reed–Solomon (RS) oracle reduction is a reduction from some relation to membership in the Reed–Solomon code, where additionally every oracle is a rational constraint.

Definition 4.4.6. *A* **Reed–Solomon (RS) oracle reduction** *over a domain $L \subseteq \mathbb{F}$ is an oracle reduction, over the base alphabet $\mathbb{F}$, from a relation $\mathcal{R}$ to the interleaved Reed–Solomon relation*

$$\mathcal{R}^*_{\mathrm{RS}} := \left\{ (\vec{\rho}, f) \text{ s.t. } \vec{\rho} \in (0,1]^*,\ f \colon L \to \mathbb{F} \text{ is a codeword in } \mathrm{RS}\left[L, \vec{\rho}\right] \right\}$$

where every virtual oracle output by the verifier is a rational constraint, except for a special instance $(\vec{\rho}_0, \mathbf{\Pi}_0)$, which the verifier must output. $\mathbf{\Pi}_0$, over alphabet $\mathbb{F}^{\sum_i s_i}$, is given by $\mathbf{\Pi}_0(\alpha) = (\pi_1(\alpha), \ldots, \pi_r(\alpha))$ (i.e., it is a stacking of the oracles sent by the prover).

In this work we will assume throughout that L comes a family of subgroups (of a family of fields) such that there is an encoding algorithm for the Reed–Solomon code on domain L with arithmetic complexity $O(|L| \log |L|)$.

Note that $\mathbf{\Pi}_0$ is not a rational constraint because its alphabet is not $\mathbb{F}$. Later we will also refer to the *non-interleaved* Reed–Solomon relation $\mathcal{R}_{\mathrm{RS}} := \{(\rho, f) : \rho \in (0, 1], f \in \mathrm{RS}\,[L, \rho]\}$.

RS oracle reductions have a useful property: if the soundness condition holds for $\delta = 0$, then the soundness condition also holds for a distance $\delta > 0$ related to the *maximum rate* of the reduction. Informally, the maximum rate is the maximum over the (prescribed) rates of codewords sent by the prover and those induced by the verifier's rational constraints. To define it, we need the notation for the *degree* and *rate* of a circuit from Definition 2.3.6.

An oracle reduction has ***maximum rate*** ρ^* if, for every rational constraint $(\sigma, \mathbf{\Pi})$ output by the verifier, $\max(\mathrm{rate}(N; \vec{\rho}_0), \sigma + \mathrm{rate}(D)) \leq \rho^*$. This expression is motivated by the proof of the following lemma; see Theorem 3.7.1 for details.

Lemma 4.4.7. *Suppose that an RS oracle reduction with maximum rate ρ^* satisfies the following* **weak soundness** *condition: if $\mathtt{x} \notin \mathcal{L}(\mathcal{R})$ then for any prover $\tilde{P}$, with probability $1 - \epsilon$ over the verifier's randomness, there exists $j \in [m]$ such that $(\rho^{(j)}, \mathbf{\Pi}^{(j)}[\pi_1, \ldots, \pi_n]) \notin \mathcal{R}_{\mathrm{RS}}$. Then the reduction satisfies the standard soundness condition (see Definition 4.4.2) with soundness error ϵ and distance $\delta := \frac{1}{2}(1 - \rho^*)$.*

This means that for the oracle reductions in this paper *we need only establish weak soundness*. Also, one can see that RS oracle reductions have locality r (the number of rounds), since $|Q(\alpha)| = r$ for all $\alpha \in L$.

The following lemma shows that, for RS oracle reductions, it suffices to run the proximity test on a single virtual oracle. This reduces the query complexity and proof length when we apply Lemma 4.4.4.

Lemma 4.4.8. *Suppose that there exists an r-round RS oracle reduction from $\mathcal{R}$ over domain L, m virtual oracles, soundness error ϵ, maximum rate ρ^*, and distance δ. Then there is an r-round oracle reduction from $\mathcal{R}$ to the* non-interleaved *Reed–Solomon relation $\mathcal{R}_{\mathrm{RS}}$ with locality r,* **one** *virtual oracle, soundness error $\epsilon + |L|/|\mathbb{F}|$, maximum rate ρ^*, and distance $\min(\delta, (1 - \rho^*)/3, (1 - 2\rho^*)/2)$.*

Proof. Implicit in Theorem 3.7.1, where it follows from [39]. □

Combining Lemmas 4.4.4, 4.4.7 and 4.4.8 yields the following useful corollary. We shall invoke it, in Section 4.9, on the two main building blocks obtained in this paper in order to prove our main result.

Corollary 4.4.9. *Suppose that there exist:*

(i) an r-round RS oracle reduction from $\mathcal{R}$ over domain L, m virtual oracles, length s and rate ρ^* that satisfies the weak soundness condition with soundness error ϵ;

(ii) an r'-round IOP of proximity for $\mathcal{R}_{\mathrm{RS}}$ with soundness error ϵ', proximity parameter $\delta' < \min((1 - \rho^*)/3, (1 - 2\rho^*)/2)$, length p and query complexity $(\mathsf{q}_{\mathsf{w}}, \mathsf{q}_{\pi})$.

Then there exists an $(r + r')$-round IOP for $\mathcal{R}$ with soundness error $\epsilon + \epsilon' + \frac{|L|}{|\mathbb{F}|}$, length $s + \mathsf{p}$ and query complexity $\mathsf{q}_{\mathsf{w}} \cdot r + \mathsf{q}_{\pi}$.

4.5 Trace embeddings

The IOPs that we construct rely on the algebraic structure of univariate polynomials. For example, all prover messages are (allegedly) Reed–Solomon codewords. Since our proof systems argue the validity of execution traces, which are two-dimensional "tall and skinny" tables representing the state of a small number of registers in each step of a long computation, we need to embed such traces into univariate polynomials, in a way that allows us to efficiently reconstruct the two-dimensional structure and traverse it. Ad-hoc methods for obtaining such embeddings are ubiquitous in the PCP literature, and typically rely on the algebraic structure of the underlying field; see, e.g., [126, 102, 41, 36, 28, 70] for examples.

We introduce *trace embeddings*, a notion that abstracts the foregoing methods and encapsulates the algebraic structure required to instantiate them. Throughout, by "efficient" we mean a function that can be evaluated in time that is polylogarithmic in the size of its domain (which is the usual notion of efficiency given an appropriate encoding of the input). Informally, a trace embedding has two components:

1. A *bivariate embedding*, which encodes a table $A \in \mathbb{F}^{N \times n}$ into a function $f_A \colon H \to \mathbb{F}$, with $H \subseteq \mathbb{F}$ of size Nn, whose (univariate) low-degree extension admits efficient extraction of A's row/column structure.

   In more detail, a bivariate embedding (Definition 4.5.3) is a method of embedding two-dimensional data, indexed by coordinates $(h_1, h_2) \in H_1 \times H_2$ (with $H_1, H_2 \subseteq \mathbb{F}$), into one-dimensional data, indexed by a single coordinate $h \in H$ (with $H \subseteq \mathbb{F}$), such that (h_1, h_2) can be efficiently derived from h. Also, the embedding's projections to its two coordinates have efficiently-computable low-degree extensions.

2. A *successor ordering*, which induces a total order on the domain and allows for efficiently moving from each element to its successor. In more detail, we equip H_1 with an ordering $\gamma \colon [|H_1|] \to H_1$ given by a "first" element $\mathbf{1}_{H_1} \in H_1$ and an efficiently-computable low-degree polynomial N. The ordering is $\gamma(1) := \mathbf{1}_{H_1}$ and $\gamma(i+1) := N(\gamma(i))$ for every $i \in \{1, \ldots, |H_1| - 1\}$.

Note the asymmetry in the definition: we require H_1 to have a successor ordering, whereas we require nothing of H_2. This is because in our application H_1 will be large (roughly the length of the computation) whereas H_2 will be small (roughly the number of registers in the computation).

For convenience, we first present the formal definition of trace embeddings below, and discuss the components it uses, as well as their properties, in the following subsections.

Definition 4.5.1. *Let $\mathbb{F}$ be a finite field. A* **trace embedding** *is a tuple $\mathcal{T} = (\Phi, \mathcal{O}, \gamma)$ where $\Phi \colon H \to H_1 \times H_2$ is an efficient bivariate embedding in $\mathbb{F}$ (see Section 4.5.1), $\mathcal{O}$ an efficient successor ordering on H_1 (see Section 4.5.2), and $\gamma \colon [|H_2|] \to H_2$ an ordering on H_2. Additionally, we require that the vanishing polynomial $\mathbb{Z}_{H_1}$ and its (standard formal) derivative can be evaluated anywhere on $\mathbb{F}$ in $\mathrm{polylog}(|H|)$ field operations.*

Letting $N := |H_1|$ and $n := |H_2|$, a trace embedding $\mathcal{T}$ induces a bijection $\mathcal{T} \colon [N] \times [n] \to H$ defined as $\mathcal{T}(i,j) := \Phi^{-1}(\gamma_{\mathcal{O}}(i), \gamma(j))$ for every $i \in [N]$ and $j \in [n]$, and $\gamma_{\mathcal{O}}$ being the

ordering induced by $\mathcal{O}$. In light of this, we sometimes simply write $\mathcal{T}\colon [N] \times [n] \to H$ to refer to a trace embedding (instead of writing the tuple that defines it), which means that we consider *any* trace embedding that induces such a bijection.

In the remainder of this section, we give the formal definitions of bivariate embeddings and successor orderings, and prove the following lemma about the existence and efficient realizability of trace embeddings.

Lemma 4.5.2. *The field $\mathbb{F}$ of size p^e, for prime p, has trace embeddings of all sizes (N, n) for which either:*

(i) Nn divides $p^e - 1$, and N, n are coprime; or
(ii) $N = p^i$ and $n = p^j$ for all i, j such that $i + j < e$.

Moreover, there is a polynomial-time algorithm that, on input a description of $\mathbb{F}$, integers N and 1^n, outputs a description of a trace embedding of size $N \times n$, if N, n satisfy one of the above conditions (and $\perp$ otherwise).

4.5.1 Bivariate embeddings

We define the notion of an (efficient) bivariate embedding, and show that it can be instantiated via the additive or multiplicative subgroup structure of a finite field. Then in Section 4.5.1.1, we state some simple linear-algebraic implications of bivariate embeddings that are used frequently in subsequent sections.

Definition 4.5.3. *Let $\mathbb{F}$ be a finite field. A (low-degree)* **bivariate embedding** *in $\mathbb{F}$ is a tuple $(\Phi_1, \Phi_2, H, H_1, H_2)$ where H, H_1, H_2 are subsets of $\mathbb{F}$, $\Phi_1 \in \mathbb{F}[X]$ is a polynomial of degree $|H_2|$, $\Phi_2 \in \mathbb{F}[X]$ is a polynomial of degree $|H_1|$, such that the function $\Phi\colon H \to H_1 \times H_2$ defined as $\Phi(h) := (\Phi_1(h), \Phi_2(h))$ is a bijection.*

A bivariate embedding $\Phi\colon H \to H_1 \times H_2$ is **efficient** *if Φ_1 and Φ_2 can be evaluated at any $\alpha \in \mathbb{F}$ in $\mathrm{polylog}(|H|)$ field operations.*

For notational convenience we refer to a bivariate embedding $(\Phi_1, \Phi_2, H, H_1, H_2)$ with the notation $\Phi\colon H \to H_1 \times H_2$, defined as in Definition 4.5.3, leaving the polynomials Φ_1, Φ_2 implicit.

The degrees of Φ_1, Φ_2 will impact the efficiency of our proof system, and so we aim to minimize them. In particular, the degree choices in the definition are *minimal*: if Φ_1 has degree $d > 0$ then $\mathbb{Z}_{H_1}(\Phi_1(X))$ has degree $|H_1| \cdot d$, is not the zero polynomial, and is zero everywhere on H, so $d \geq |H|/|H_1| = |H_2|$; a similar statement holds for Φ_2. Since we achieve these degree bounds in the constructions below, we make these choices part of the above definition. In particular, $\mathbb{Z}_{H_1}(\Phi_1(X))$ and $\mathbb{Z}_{H_2}(\Phi_2(X))$ are each multiples of $\mathbb{Z}_H$.

While for any $H, H_1, H_2 \subseteq \mathbb{F}$ with $|H| = |H_1| \cdot |H_2|$ there exist (many) bijections $H \to H_1 \times H_2$, the aforementioned degree constraints severely limit our choices for these sets. All known constructions use the group structure of H. It remains an intriguing open question to determine whether other constructions exist.

Efficiency is an even stronger requirement. Since the "truth table" of Φ is of size $|H|$, it must be that H has some inherent product structure that we can exploit. In our protocols, H will

be an additive or multiplicative subgroup of $\mathbb{F}$ and H_1, H_2 will be isomorphic to subgroups of H whose product is H.

We now construct efficient bivariate embeddings, over multiplicative and additive subgroups of the field.

Lemma 4.5.4. *Let $\mathbb{F}$ be a finite field, and let H, H_1, H_2 be* **multiplicative** *subgroups of $\mathbb{F}$ such that $|H| = |H_1| \cdot |H_2|$ and $\gcd(|H_1|, |H_2|) = 1$. Then, there exists an efficient bivariate embedding $\Phi\colon H \to H_1 \times H_2$.*

Proof. Map each $h \in H$ to the pair $\Phi(h) := (h^{|H_2|}, h^{|H_1|}) \in H_1 \times H_2$. By the Chinese Remainder Theorem, Φ is an isomorphism. Moreover, the polynomials $\Phi_1(X) := X^{|H_2|}$ and $\Phi_2(X) := X^{|H_1|}$ agree with Φ on H, and can be evaluated in $O(\log(|H_1|) + \log(|H_2|)) = O(\log(|H|))$ field operations. □

Lemma 4.5.5. *Let $\mathbb{F}$ be a finite field, and let V, W be* **additive** *subgroups of $\mathbb{F}$ with respective sizes m, n and such that $V \cap W = \{0\}$. Then there exists an efficient bivariate embedding $\Phi\colon V \oplus W \to \mathbb{Z}_W(V) \times \mathbb{Z}_V(W)$.*

Proof. Map each $v + w \in V \oplus W$ (with $v \in V$ and $w \in W$) to $\Phi(v + w) := (\mathbb{Z}_W(v), \mathbb{Z}_V(w)) \in \mathbb{Z}_W(V) \times \mathbb{Z}_V(W)$, where $\mathbb{Z}_W$ and $\mathbb{Z}_V$ are the vanishing polynomials of V and W. Since $\mathbb{Z}_W$ is injective on V and $\mathbb{Z}_V$ is injective on W, Φ is a bijection. Moreover, the polynomials $\Phi_1(X) := \mathbb{Z}_W(X)$ and $\Phi_2(X) := \mathbb{Z}_V(X)$ can be evaluated in $O(\log^2(|W|) + \log^2(|V|)) = O(\log^2(|H|))$ field operations. □

4.5.1.1 Linear-algebraic properties

We use bivariate embeddings as a natural, and algebraically friendly, way to identify the tensor product space $\mathbb{F}^{H_1} \otimes \mathbb{F}^{H_2}$ with $\mathbb{F}^H$. This is expressed via the definitions and propositions below, which we state without proof. The propositions follow from standard linear algebra and the fact that a bivariate embedding extends bijections $\gamma_1\colon H_1 \to [|H_1|]$ and $\gamma_2\colon H_2 \to [|H_2|]$ to a bijection $\gamma\colon H \to [|H_1|] \times [|H_2|]$.

Definition 4.5.6. *Define $\otimes_\Phi\colon \mathbb{F}^{H_1} \times \mathbb{F}^{H_2} \to \mathbb{F}^H$ to be the bilinear function that maps $u \in \mathbb{F}^{H_1}$ and $v \in \mathbb{F}^{H_2}$ to $u \otimes_\Phi v \in \mathbb{F}^H$ where $(u \otimes_\Phi v)(h) := u(\Phi_1(h)) \cdot v(\Phi_2(h))$ for every $h \in H$.*

Proposition 4.5.7. *The function $\otimes_\Phi\colon \mathbb{F}^{H_1} \times \mathbb{F}^{H_2} \to \mathbb{F}^H$ is a tensor product.*

Definition 4.5.8. *Let $\Phi\colon H \to H_1 \times H_2$ be a bivariate embedding. The* **Kronecker product** *of matrices $A \in \mathbb{F}^{H_1 \times H_1}$ and $B \in \mathbb{F}^{H_2 \times H_2}$ with respect to Φ is the matrix $A \otimes_\Phi B \in \mathbb{F}^{H \times H}$ where $(A \otimes_\Phi B)(h, h') := A(\Phi_1(h), \Phi_1(h')) \cdot B(\Phi_2(h), \Phi_2(h'))$ for every $h, h' \in H$.*

Proposition 4.5.9. *Let $A \in \mathbb{F}^{H_1 \times H_1}$ and $B \in \mathbb{F}^{H_2 \times H_2}$. Then $(A \otimes_\Phi B)$ is the unique linear map such that, for every $u \in \mathbb{F}^{H_1}$ and $v \in \mathbb{F}^{H_2}$, $(A \otimes_\Phi B)(u \otimes_\Phi v) = (Au) \otimes_\Phi (Bv)$, under usual matrix multiplication.*

4.5.2 Successor orderings

A *successor ordering* of a set S is a pair $\mathcal{O} = (\mathbf{1}_S, N)$ where $\mathbf{1}_S \in S$ is a distinguished first element and $N \in \mathbb{F}[X]$ is a degree-1 *successor polynomial* that induces an ordering by mapping each element of S to its "successor". That is, $\mathcal{O}$ induces a bijection $\gamma_{\mathcal{O}} \colon [|S|] \to S$ given by $\gamma_{\mathcal{O}}(1) = \mathbf{1}_S$, $\gamma_{\mathcal{O}}(2) = N(\mathbf{1}_S)$, $\gamma_{\mathcal{O}}(3) = N(N(\mathbf{1}_S))$, and so on. We call $\mathcal{O}$ *efficient* if N can be evaluated in $\mathrm{polylog}(|S|)$ field operations.

We show that all multiplicative subgroups S of $\mathbb{F}$ have efficient successor orderings, by relying on any isomorphism $S \cong \mathbb{Z}_{|S|}$. We also show that, for $|\mathbb{F}| = p^e$, there exists an additive subgroup S of $\mathbb{F}$ of size p^i with an efficient successor ordering, if p is small enough. This relies on "simulating" the field $\mathbb{F}_{p^i}$ inside S.

4.5.2.1 Multiplicative subgroups

The construction for multiplicative subgroups is straightforward. The following lemma gives the construction of a successor ordering for any multiplicative subgroup of $\mathbb{F}$, that is, of any size dividing $|\mathbb{F}| - 1$.

Lemma 4.5.10. *Let $\mathbb{F}$ be a finite field. Every multiplicative subgroup S of $\mathbb{F}$ has an efficient successor ordering.*

Proof. Choose a generator g of S (note that S is cyclic), and then let $\mathbf{1}_S := 1_{\mathbb{F}}$ and $N(X) := gX$. □

4.5.2.2 Additive subgroups

In order to give the construction for additive subgroups, we first need to generalize the definition of successor ordering given above. That is, we need to be more permissive about the successor polynomials N we allow.

In later sections, we use successor orderings by composing the polynomial N with other polynomials that enforce correct computation. We must ensure that this composition does not have too large of a degree. Ideally we would like N to have degree $O(1)$ because then $\deg(g \circ N) = O(\deg g)$ for any $g \in \mathbb{F}[X]$. When S is a multiplicative subgroup we can achieve this (as N has degree 1), but when S is an additive subgroup we get $\deg(N) = \Omega(|S|)$, which would give $\deg(g \circ N) = \Omega(|S| \deg g)$. Since $|S|$ and $\deg(g)$ will be each approximately the computation length T, the degree of the composed polynomial would be $\Omega(T^2)$. This would prevent us from achieving, e.g., IOPs of linear proof length.

To deal with this, we use the fact that the additive N satisfies a useful structural property, to which we refer as being *piecewise polynomial*. A function f is a piecewise polynomial of degree d with respect to a partition $(S_1, \ldots, S_\ell)$ of S if there exist polynomials $(f_1, \ldots, f_\ell)$ of degree d such that

$$\forall\, i \in [\ell],\, \forall\, \alpha \in S_i\,,\ f(\alpha) = f_i(\alpha)\ .$$

If s_i is a degree-$|S|$ extension of the indicator for S_i in S then $\sum_{i=1}^{\ell} s_i(X) g(f_i(X))$ has degree $O(|S| + \deg g)$ and agrees with $g \circ N$ on S. This is an additive rather than multiplicative increase in the degree, and later will yield a degree bound of $O(T)$ as required. We now define formally the notion of a piecewise polynomial.

Definition 4.5.11. *Let $\mathbb{F}$ be a finite field and $S \subseteq \mathbb{F}$. A* **piecewise polynomial on** *S is a pair $F = (\mathcal{S}, \mathcal{F})$ where $\mathcal{S} = (S_1, \ldots, S_\ell)$ is a partition of S and $\mathcal{F} = (f_1, \ldots, f_\ell) \in \mathbb{F}[X]^\ell$. Let s_i be the unique extension of degree less than $|S|$ of the indicator for S_i in S. The* **value** *of $F = (\mathcal{S}, \mathcal{F})$ at $\alpha \in \mathbb{F}$ is $F(\alpha) := \sum_{i=1}^{\ell} s_i(\alpha) f_i(\alpha)$.*

A piecewise polynomial $(\mathcal{S}, \mathcal{F})$ has **piecewise degree** *d if $\max_{f \in \mathcal{F}} \deg(f) \leq d$. A piecewise polynomial is* **efficient** *if each s_i, f_i can be evaluated in time $\mathrm{polylog}(|S|)$, and $\ell = \mathrm{polylog}(|S|)$.*

The following proposition shows that the special structure of piecewise polynomials allows for composition with an additive, rather than multiplicative, dependence on $|S|$.

Proposition 4.5.12. *If $F = (\mathcal{S}, \mathcal{F})$ is piecewise polynomial on S with degree d and $g \in \mathbb{F}[X]$, then there is a polynomial h of degree $|S| + d \cdot \deg(g)$ that agrees with $g \circ F$ on S. Moreover, if F is efficient and g can be evaluated in $\mathrm{polylog}(|S|)$ field operations then h can be evaluated in $\mathrm{polylog}(|S|)$ field operations.*

Proof. Let $h(X) := \sum_{i=1}^{\ell} s_i(X) g(f_i(X)) \in \mathbb{F}[X]$, where s_i is the unique extension of degree less than $|S|$ of the indicator function of S_i in S. Then since the s_i are indicators for a partition of S,

$$\forall\, \alpha \in S\,,\ g(f(\alpha)) = g\left(\sum_{i=1}^{\ell} s_i(\alpha) f_i(\alpha)\right) = \sum_{i=1}^{\ell} s_i(\alpha) g(f_i(\alpha)) = h(\alpha)\ .$$

Observe that h has the required degree and can be evaluated in $\mathrm{polylog}(|S|)$ field operations. □

Now we are ready to formally define a successor ordering.

Definition 4.5.13. *Let $\mathbb{F}$ be a finite field and $S \subseteq \mathbb{F}$. A* **successor ordering** *on S is a pair $\mathcal{O} = (\mathbf{1}_S, N)$ where $\mathbf{1}_S \in S$ and N is a piecewise polynomial on S of degree 1 such that $S = \{\alpha_1, \ldots, \alpha_{|S|}\}$, where*

$$\alpha_1 := \mathbf{1}_S \text{ and } \alpha_{i+1} := N(\alpha_i) \text{ inductively for every } i \in \{1, \ldots, |S| - 1\}.$$

Let $\gamma_{\mathcal{O}} \colon S \to \{1, \ldots, |S|\}$ be the ordering on S induced by $\mathcal{O}$, i.e., $\gamma_{\mathcal{O}}(\alpha_i) := i$ for every $i \in \{1, \ldots, |S|\}$.

A successor ordering is **efficient** *if the piecewise polynomial N is efficient.*

Note that since a polynomial of degree d has a trivial piecewise representation of degree d, the foregoing definition also captures the simpler multiplicative case.

Let p be prime. For the field $\mathbb{F}$ of size p^e, we give constructions of efficient ordered subsets of any size p^i (for $1 \leq i < e$) provided p is small enough relative to i.

The following lemma appears in [41, 28, 23]. We restate it here in the language of successor orderings, and for completeness we also provide a proof.

Lemma 4.5.14. *Let $\mathbb{F}$ be a finite field of size p^e. For every $i \in \{1, \ldots, e-1\}$ there exists an additive subgroup S in $\mathbb{F}$ of size p^i with a successor ordering $\mathcal{O}$. If in addition $p = O(\log |S|)$ then $\mathcal{O}$ is efficient.*

Proof. We view $\mathbb{F}$ as $\mathbb{F}_p[\xi]/(f)$ for some $f \in \mathbb{F}_p[\xi]$. To prove the lemma, we first move to the ring $\mathbb{F}_p[\xi]$ and show that there exists a subspace of $\mathbb{F}_p[\xi]$ that is isomorphic to a "multiplicative" subset of $\mathbb{F}_p[\xi]$.

Claim 4.5.15. *Let* $S := \mathrm{span}\{1, \xi, \ldots, \xi^{i-1}\} \subseteq \mathbb{F}_p[\xi]$*, and let* $G := \{0, 1, \xi, \xi^2, \ldots, \xi^{p^i-2}\} \subseteq \mathbb{F}_p[\xi]$*. Let* $g(\xi) \in \mathbb{F}_p[\xi]$ *be a primitive polynomial of degree* i*, and let* $\varphi\colon \mathbb{F}_p[\xi] \to \mathbb{F}_p[\xi]$ *be defined as* $\varphi(f) := f \bmod g$ *where division is in the ring* $\mathbb{F}_p[\xi]$*. Then* $S = \varphi(G)$*.*

Proof of claim. It suffices to show that φ is a bijection on G, since S is the image of φ (i.e., $S = \varphi(\mathbb{F}_p[\xi])$). Since g is primitive, g has a root $\omega \in \mathbb{F}_{p^i}$ such that $\mathbb{F}_{p^i} = \{0, 1, \omega, \omega^2, \ldots, \omega^{p^i-2}\}$. Since $\mathbb{F}_p[\xi]/(g(\xi))$ is isomorphic to $\mathbb{F}_{p^i}$ via the map $\xi \mapsto \omega$, we deduce that $|\varphi(G)| = |G|$. Thus $S = \varphi(G)$ as claimed. □

We have that S is isomorphic to $S + (f) \subseteq \mathbb{F}$, and so we can regard S as a subset of $\mathbb{F}$. We now construct a successor ordering $\mathcal{O} = (\mathbf{1}_S, N)$ on S. We set $\mathbf{1}_S := 0$, and are then left to define the piecewise polynomial N. We divide S into cosets of $S' := \mathrm{span}\{1, \xi, \ldots, \xi^{i-2}\}$ as $S = \bigcup_{c \in \mathbb{F}_p} c\xi^{i-1} + S'$. These cosets have the property that if $\varphi(\xi^j) \in c\xi^{i-1} + S'$ then $\varphi(\xi^{j+1}) = \xi \cdot \varphi(\xi^j) - c \cdot g(\xi)$. Note that this is true in $\mathbb{F}$ since it is true in $\mathbb{F}_p[\xi]$ and $\deg(f) > i = \deg(g)$.

Our partition of S consists of the sets $\{0\}$, $S' \setminus \{0\}$, and the set $S' + c\xi^{i-1}$ for all $c \in \mathbb{F}_p \setminus \{0\}$. The corresponding polynomial partition is $\{L_{S,0}, L_0(\mathbb{Z}_{S'}(X)) - L_{S,0}(X)\} \cup \{L_c \circ \mathbb{Z}_{S'}\}_{c \in \mathbb{F}_p}$, where

- $L_{S,0}$ is the unique polynomial of degree less than $|S|$ with $L_{S,0}(0) = 1$ and $L_{S,0}(\gamma) = 0$ for $\gamma \in S \setminus \{0\}$;
- L_c is the unique polynomial of degree less than p such that

$$L_c(c \cdot \mathbb{Z}_{S'}(\xi^{i-1})) = 1 \quad \text{and} \quad L_c(\gamma \cdot \mathbb{Z}_{S'}(\xi^{i-1})) = 0 \text{ for every } \gamma \in \mathbb{F}_p \setminus \{c\} \ .$$

By the $\mathbb{F}_p$-linearity of $\mathbb{Z}_{S'}$, we have that, for every $\alpha \in S$, $L_c(\mathbb{Z}_{S'}(\alpha)) = 1$ if $\alpha \in S' + c\xi^{i-1}$ and 0 otherwise. Hence this is indeed the desired partition. The value of N on $a \in S$ should be

$$N(a) = \begin{cases} 1 & \text{if } a = 0 \\ \xi \cdot a & \text{if } a \in S' \setminus \{0\} \\ \xi \cdot a - c \cdot g(\xi) & \text{if } a \in S' + c\xi^{i-1} \end{cases}$$

This gives $N(0) = 1 = \varphi(1)$, and $N(\varphi(\xi^j)) = \varphi(\xi^{j+1})$ for every $j \in \{0, 1, \ldots, p^i - 2\}$.

Thus, the polynomial

$$N(X) := L_{S,0}(X) \cdot 1 + (L_0(\mathbb{Z}_{S'}(X)) - L_{S,0}(X)) \cdot \xi \cdot X + \sum_{c \in \mathbb{F}_p \setminus \{0\}} L_c(\mathbb{Z}_{S'}(X)) \cdot (\xi \cdot X - c \cdot g(\xi))$$

takes the correct values on S, and has piecewise degree 1.

Finally, observe that N can be evaluated in $\mathrm{poly}(p, \log(|S|))$ field operations, which is $\mathrm{polylog}(|S|))$ when $p = O(\log |S|)$. Indeed: (1) since S is a subspace, a Lagrange polynomial $L_{S,0}$ can be evaluated in $\mathrm{polylog}(|S|)$ field operations; (2) L_c is a Lagrange polynomial over the field of size p and thus can be evaluated in $O(\log p)$ field operations; and (3) since S' is a subspace, the vanishing polynomial $\mathbb{Z}_{S'}$ can be evaluated in $\mathrm{polylog}(|S'|)$ field operations. The dependence on p (rather than $\log p$) in the cost for evaluating N comes from adding the terms in the sum. □

4.6 A succinct lincheck protocol

We describe *succinct lincheck*, an oracle reduction for checking a useful class of succinctly-represented linear relations on Reed–Solomon codewords. This oracle reduction is later used to obtain an oracle reduction for R1CS automata (see Section 4.7), and can be viewed as a succinct analogue of the univariate lincheck protocol (Section 3.5). We recall the *lincheck relation* defined in that section, and for simplicity focus on the special case of linear relations defined by *square* constraint matrices.

Definition 4.6.1. *The* **lincheck relation** $\mathcal{R}_{\text{LIN}}$ *consists of pairs* $\big((\mathbb{F}, L, H, \rho, M), (f_1, f_2)\big)$ *where* $\mathbb{F}$ *is a finite field,* $L, H \subseteq \mathbb{F}$*,* $\rho \in (0, 1)$*,* $f_1, f_2 \in \mathrm{RS}\,[L, \rho]$*,* $M \in \mathbb{F}^{H \times H}$*, and* $\forall\, a \in H\ \ \hat{f}_1(a) = \sum_{b \in H} M_{a,b} \cdot \hat{f}_2(b)$*.*

Recall that the protocol for $\mathcal{R}_{\text{LIN}}$ in Section 3.5, which we outlined in Section 4.2.2, supports *any* constraint matrix $M \in \mathbb{F}^{H \times H}$, and so it cannot run in time $o(||M||)$ (let alone in our target time of polylogarithmic in $||M||$) for *every* matrix M, because the verifier must at least *read* the description of M.

In this section, we show that, for an expressive family of constraint matrices, we can design a "succinct" lincheck protocol wherein the verifier runs *exponentially* faster than in Protocol 3.5.3 To this end, we first observe that the verifier's expensive work in the lincheck protocol consists of evaluating low-degree extensions of the two vectors $r_\alpha := (g_h(\alpha))_{h \in H} \in \mathbb{F}^H$ and $s_\alpha := r_\alpha M \in \mathbb{F}^H$; this involves $\Omega(||M|| + |H|)$ field operations. Then we show how to perform such evaluations in $\mathrm{polylog}(|H|)$ field operations (and thereby make the verifier exponentially faster) for a class of matrices M that arise from T-time computations. Informally, we rely on a clever choice of linearly-independent polynomials $(g_h(\alpha))_{h \in H}$ and also on special algebraic properties of the matrices M, related to the algebraic structure of $\mathbb{F}$.

We now motivate the algebraic properties that we use, and then state our result.

Semisuccinct matrices. The lincheck protocol (Protocol 3.5.3) chooses the linearly independent polynomials $(g_h(X))_{h \in H}$ to be the *standard basis* $(1, X, X^2, \ldots, X^{|H|-1})$. We do not know how to efficiently evaluate the low-degree extension of r_α with respect to this basis. But there is another natural choice of basis for polynomials, the *Lagrange basis* $(L_{H,h})_{h \in H}$. In Section 4.6.1 we show that the low-degree extension $\hat{r}_\alpha$ of r_α with respect to this basis has a simple form that allows one to evaluate $\hat{r}_\alpha$ in time $\mathrm{polylog}(|H|)$.

The foregoing suggests a natural condition on the matrix M to require: if we can efficiently compute a low-degree extension of a vector $v \in \mathbb{F}^H$, then we should also be able to efficiently compute a low-degree extension of the vector $vM \in \mathbb{F}^H$. This notion of (algebraic) *succinctness* is formally captured as follows.

Definition 4.6.2. *Let* $H \subseteq \mathbb{F}$ *and* $\kappa \colon \mathbb{N} \to \mathbb{N}$*. A vector* $v \in \mathbb{F}^H$ *is* d**-extendable** *if there is* $p \in \mathbb{F}[X]$ *of degree at most* d *that agrees with* v *on* H *and* p *can be evaluated at any* $\alpha \in \mathbb{F}$ *in* $\mathrm{polylog}(|H|)$ *field operations. A matrix* $A \in \mathbb{F}^{H \times H}$ *is* κ**-succinct** *if, for every* $d \geq |H| - 1$*,* vA *is* $\kappa(d)$*-extendable whenever* v *is* d*-extendable.*

The identity matrix trivially satisfies the above definition, and later on we will show that the matrix with 1s on the superdiagonal is also algebraically succinct for certain choices of $\mathbb{F}$ and H (see Lemma 4.7.4).

Unfortunately, Definition 4.6.2 turns out to be too restrictive for us. But it is a starting point for a more general notion that suffices. Concretely, the matrices that arise in Section 4.7 are not themselves succinct, but they can be *decomposed* into a part that is succinct and a part that is "small". This is analogous to a Turing machine, where a computation is specified by repeated applications of a small transition function.

Definition 4.6.3. *Let $\Phi\colon H \to H_1 \times H_2$ be a bivariate embedding, and let $\kappa\colon \mathbb{N} \to \mathbb{N}$. A matrix $M \in \mathbb{F}^{H\times H}$ is* (Φ,κ)**-semisuccinct** *if $M = A \otimes_\Phi B$ for κ-succinct $A \in \mathbb{F}^{H_1\times H_1}$ and arbitrary $B \in \mathbb{F}^{H_2\times H_2}$.*

In the definition there is an asymmetry between H_1 and H_2, since we think of H_1 as large and H_2 as small.

To handle semisuccinct matrices we need a slightly different property from the polynomials $(g_h(X))_{h\in H}$. Namely we need that the vector $r_\alpha := (g_h(\alpha))_{h\in H}$ can be written as a *tensor product* of vectors $r_\alpha^{(1)} \in \mathbb{F}^{H_1}$ and $r_\alpha^{(2)} \in \mathbb{F}^{H_2}$ such that we can efficiently compute the low-degree extension of $r_\alpha^{(1)}$. Then, by definition of the Kronecker product, $r_\alpha M = (r_\alpha^{(1)} A) \otimes (r_\alpha^{(2)} B)$. Since A is succinct and B is small, we can compute the LDEs of $r_\alpha^{(1)} A$ and $r_\alpha^{(2)} B$ efficiently, then use Lemma 4.6.7 to compute the LDE of their tensor product. In Section 4.6.2 we show how to construct $(g_h(X))_{h\in H}$ from the Lagrange bases on H_1 and H_2.

A succinct lincheck protocol. The main result of the section is an oracle reduction, with linear length and locality 2, from lincheck on sums of semisuccinct matrices. The verifier uses $\mathrm{poly}(|\mathbb{x}|) = \mathrm{poly}(\ell, n, \log N)$ field operations, which is *polylogarithmic* in the size of the succinct part of the matrix.

Lemma 4.6.4. *Let $\mathbb{F}$ be a finite field with an efficient bivariate embedding $\Phi\colon H \to H_1 \times H_2$ where H is a coset in $\mathbb{F}$. Suppose that $M \in \mathbb{F}^{H\times H}$ has the form $M = \sum_{i=1}^{\ell} M_i$ where each $M_i \in \mathbb{F}^{H\times H}$ is (Φ,κ)-semisuccinct.*

Setting $N := |H_1|$, $n := |H_2|$, and $d := n \cdot (N + \kappa(N))$, there is a 1-round RS oracle reduction of proximity (Protocol 4.6.12) for $\mathcal{R}_{\mathrm{LIN}}$ for instances of the form $\mathbb{x} = (\mathbb{F}, L, H, \rho, M)$ with the following parameters:

length	$\lvert L\rvert$	*soundness error*	$d/\lvert\mathbb{F}\rvert$	*prover time*	$\lvert L\rvert \log \lvert L\rvert + \lvert\lvert M\rvert\rvert$
locality	2	*distance*	$\frac{1}{2}(1-\rho-\frac{d}{\lvert L\rvert})$	*verifier time*	$\mathrm{poly}(\lvert\mathbb{x}\rvert)$

(Above $||M||$ denotes the number of non-zero entries in the matrix $M \in \mathbb{F}^{H\times H}$.)

Organization. In Sections 4.6.1 and 4.6.2 we develop algebraic preliminaries. In Section 4.6.3 we prove Lemma 4.6.4. In Section 4.6.4 we extend Lemma 4.6.4 to handle a *block-matrix* lincheck relation. This latter relation is the one that we actually use in Section 4.7 to obtain an oracle reduction for interactive R1CS automata.

4.6.1 Properties of the Lagrange basis

Let $\mathbb{F}$ be a finite field and S a subset of $\mathbb{F}$. The *Lagrange basis* over S are the polynomials $L_S := (L_{S,\alpha})_{\alpha\in S}$ where $L_{S,\alpha}$ is the unique polynomial of degree less than $|S|$ with $L_{S,\alpha}(\alpha) = 1$

$$\begin{array}{ccc} \hat{r}_S(f(X), Y) & \xrightarrow{\mathsf{eval}_Y(S)} & \mathbf{r}_S^f \\ {\scriptstyle \mathsf{eval}_X(\alpha)}\downarrow & & \downarrow{\scriptstyle \mathsf{eval}'_X(\alpha)} \\ \hat{r}_S(f(\alpha), Y) & \xleftarrow{\mathsf{LDE}} & \mathbf{r}_S^{f(\alpha)} \end{array}$$

Figure 4.3: Commutative diagram showing the relationship between the vector of (univariate) polynomials $\mathbf{r}_S$ and the bivariate polynomial $\hat{r}_S$. The function $\mathsf{eval}_Y(S)$ maps a polynomial $g(X, Y)$ to the vector of polynomials $(g(X, \beta))_{\beta \in S}$. The function $\mathsf{eval}_X(\alpha)$ maps a polynomial $g(X, Y)$ to the polynomial $g(\alpha, Y)$. The function $\mathsf{eval}'_X(\alpha)$ maps a vector of polynomials $(g_\beta(X))_{\beta \in S}$ to the vector of polynomials $(g_\beta(\alpha))_{\alpha \in S}$. The function LDE maps a vector $(r_\beta)_{\beta \in S} \in \mathbb{F}^S$ to the unique polynomial g of degree less than $|S|$ such that $g(\beta) = r_\beta$ for all $\beta \in S$.

and $L_{S,\alpha}(\gamma) = 0$ for all $\gamma \in S \setminus \{\alpha\}$. Recall that L_S is a basis for the (vector space of) polynomials in $\mathbb{F}[X]$ of degree less than $|S|$.

In this work it will be convenient to consider an *unnormalized* version of the Lagrange basis. We let $L'_{S,\beta} := \mathbb{Z}_S(X)/(X - \beta)$, and define the vector of polynomials

$$\mathbf{r}_S := (L'_{S,\beta})_{\beta \in S} = \left(\frac{\mathbb{Z}_S(X)}{X - \beta}\right)_{\beta \in S} \in \mathbb{F}[X]^S \ .$$

Observe that $L'_{S,\beta}$ is a polynomial of degree less than $|S|$ that is zero on $S \setminus \{\beta\}$, and is therefore equal to $L_{S,\beta}$ up to a multiplicative (nonzero) constant factor. It follows that $\mathbf{r}_S$ is also a basis for the vector space of polynomials of degree less than $|S|$. The following lemma shows that the unique low-degree extension of $\mathbf{r}_S \in \mathbb{F}[X]^S$, which is a bivariate polynomial $\hat{r}_S \in \mathbb{F}[X, Y]$, has a simple explicit form.

Lemma 4.6.5. *$\hat{r}_S(X, Y) := (\mathbb{Z}_S(X) - \mathbb{Z}_S(Y))/(X - Y)$ is the unique low-degree extension of $\mathbf{r}_S$.*

Proof. Observe that $\hat{r}_S$ is a polynomial of degree $|S| - 1$ in Y because $X - Y$ divides $\mathbb{Z}_S(X) - \mathbb{Z}_S(Y)$. Moreover, for every $\beta \in S$, $\hat{r}_S(X, \beta) = \mathbb{Z}_S(X)/(X - \beta) = L'_{S,\beta}(X)$, which is the β-th entry of $\mathbf{r}_S$. □

Given a polynomial $f \in \mathbb{F}[X]$, we define $\mathbf{r}_S^f \in \mathbb{F}[X]^S$ to be the vector of polynomials obtained by composing each entry of $\mathbf{r}_S$ with f:

$$\mathbf{r}_S^f := (L'_{S,\beta} \circ f)_{\beta \in S} = \left(\frac{\mathbb{Z}_S(f(X))}{f(X) - \beta}\right)_{\beta \in S} \in \mathbb{F}[X]^S \ .$$

Note that when $f = \alpha \in \mathbb{F}$, this corresponds to evaluating each entry of the vector $\mathbf{r}_S$ at the point α. Also, Lemma 4.6.5 implies that the unique low-degree extension of $\mathbf{r}_S^f$ is $\hat{r}_S(f(X), Y)$. The next lemma shows that for certain sets S we can evaluate this low-degree extension very efficiently.

Lemma 4.6.6. *Let S be a subset of $\mathbb{F}$ whose vanishing polynomial $\mathbb{Z}_S$ and its formal derivative $\mathrm{D}\mathbb{Z}_S$ can both be evaluated at any point in $\mathrm{polylog}(|S|)$ field operations. Then, given any $\alpha, \beta \in \mathbb{F}$, the unique low-degree extension of $\mathbf{r}_S^\alpha$, which is $\hat{r}_S(\alpha, Y)$, can be evaluated at β in $\mathrm{polylog}(|S|)$ field operations. In particular, this holds when the set S is an additive or multiplicative subgroup of $\mathbb{F}$.*

Proof. If $\alpha - \beta \neq 0$, we can evaluate $\hat{r}_S(\alpha, \beta)$ directly by computing $(\mathbb{Z}_S(\alpha) - \mathbb{Z}_S(\beta))/(\alpha - \beta)$ in $\mathrm{polylog}(|S|)$ field operations. If instead $\alpha - \beta = 0$ then we use a different approach: observing that the polynomial $\hat{r}_S(X, X)$ is the formal derivative of $\mathbb{Z}_S(X)$, we evaluate $\hat{r}_S(\alpha, \beta)$ by computing $\hat{r}_S(\beta, \beta) = (\mathrm{D}\mathbb{Z}_S)(\beta)$. (Note that when S is a multiplicative subgroup then $\mathrm{D}\mathbb{Z}_S(X) = |S|X^{|S|-1}$. And when S is an additive subgroup, $\mathbb{Z}_S$ is a linearized polynomial, so its derivative is the coefficient of the linear term.) □

4.6.2 Efficient linear independence via the tensor product

We use the bivariate polynomial introduced in Section 4.6.1 to find a vector of linearly independent polynomials that decomposes via the tensor product (see Definition 4.5.6). We begin by noting that the efficiency of evaluating low-degree extensions is preserved by tensor products.

Lemma 4.6.7. *Let $\Phi\colon H \to H_1 \times H_2$ be an efficient bivariate embedding (see Definition 4.5.3). Let $\hat{u} \in \mathbb{F}[X]$ be a degree-d_u extension of $u \in \mathbb{F}^{H_1}$, and $\hat{v} \in \mathbb{F}[X]$ a degree-d_v extension of $v \in \mathbb{F}^{H_2}$. For every $\beta \in \mathbb{F}$, if $\hat{u}, \hat{v}$ can be evaluated at β in time T_u, T_v, then an extension of $u \otimes_\Phi v \in \mathbb{F}^H$ of degree $|H_2| \cdot d_u + |H_1| \cdot d_v$ can be evaluated at β in time $O(T_u + T_v + \mathrm{polylog}\,|H|)$.*

Proof. The polynomial $\hat{u}(\Phi_1(X))\hat{v}(\Phi_2(X)) \in \mathbb{F}[X]$ agrees with the vector $u \otimes_\Phi v \in \mathbb{F}^H$ on H, has degree $|H_2| \cdot d_u + |H_1| \cdot d_v$, and can be evaluated in time $O(T_u + T_v + \mathrm{polylog}\,|H|)$. □

We define a vector of polynomials $\mathbf{t}_H^\Phi \in \mathbb{F}[X]^H$ and, for every $\alpha \in \mathbb{F}$, the vector of elements $\mathbf{t}_H^{\Phi(\alpha)} \in \mathbb{F}^H$ obtained by evaluating each polynomial. We prove that the polynomials in $\mathbf{t}_H^\Phi$ are linearly independent (Lemma 4.6.9), that a low-degree extension of $\mathbf{t}_H^{\Phi(\alpha)}$ can be efficiently evaluated (Corollary 4.6.10), and that a low-degree extension of $\mathbf{t}_H^{\Phi(\alpha)} M$ can be efficiently evaluated when M is semisuccinct (Corollary 4.6.11).

Definition 4.6.8. *Let $\Phi\colon H \to H_1 \times H_2$ be a bivariate embedding. We define $\mathbf{t}_H^\Phi \in \mathbb{F}[X]^H$ to be the vector of polynomials given by*

$$\mathbf{t}_H^\Phi := \mathbf{r}_{H_1}^{\Phi_1} \otimes_\Phi \mathbf{r}_{H_2}^{\Phi_2} = \left(L'_{H_1,h_1}(\Phi_1(X))\right)_{h_1 \in H_1} \otimes_\Phi \left(L'_{H_2,h_2}(\Phi_2(X))\right)_{h_2 \in H_2}.$$

Moreover, for any $\alpha \in \mathbb{F}$, we define $\mathbf{t}_H^{\Phi(\alpha)} \in \mathbb{F}^H$ to be the vector of field elements given by

$$\mathbf{t}_H^{\Phi(\alpha)} := \mathbf{r}_{H_1}^{\Phi_1(\alpha)} \otimes_\Phi \mathbf{r}_{H_2}^{\Phi_2(\alpha)} = \left(L'_{H_1,h_1}(\Phi_1(\alpha))\right)_{h_1 \in H_1} \otimes_\Phi \left(L'_{H_2,h_2}(\Phi_2(\alpha))\right)_{h_2 \in H_2}.$$

Lemma 4.6.9. *Let $\Phi\colon H \to H_1 \times H_2$ be a bivariate embedding. Then $\mathbf{t}_H^\Phi \in \mathbb{F}[X]^H$ is a vector of $|H|$ linearly independent polynomials of degree less than $2|H|$. (Note that $\mathbf{t}_H^\Phi$ is not a* basis *for the space of polynomials of degree less than $2|H|$ because it contains only $|H|$ elements.)*

Proof. For every $h \in H$ there exists $c \in \mathbb{F}$ such that

$$(\mathbf{r}_{H_1}^{\Phi_1} \otimes_\Phi \mathbf{r}_{H_2}^{\Phi_2})(h) = c \cdot L_{H_1,\Phi_1(h)}(\Phi_1(X)) \cdot L_{H_2,\Phi_2(h)}(\Phi_2(X)) \ .$$

This is a polynomial of degree less than $2|H_1||H_2| = 2|H|$ that is zero everywhere on H except at h. □

Corollary 4.6.10. *Let $\Phi\colon H \to H_1 \times H_2$ be an efficient bivariate embedding. For every $\alpha \in \mathbb{F}$, a degree-$2|H|$ extension of $\mathbf{t}_H^{\Phi(\alpha)} \in \mathbb{F}^H$ can be evaluated at any $\beta \in \mathbb{F}$ in $\mathrm{poly}(\log|H_1|, \log|H_2|)$ field operations.*

Proof. The polynomials Φ_1 and Φ_2 can be evaluated at α in $\mathrm{polylog}(|H|)$ field operations, since Φ is efficient. By Lemma 4.6.6, the unique low-degree extensions of $\mathbf{r}_{H_1}^{\Phi_1(\alpha)}$ and $\mathbf{r}_{H_2}^{\Phi_2(\alpha)}$ can each be evaluated at β in time $\mathrm{polylog}(|H|)$. Note that these polynomials have degrees $|H_1|$ and $|H_2|$ respectively. Thus, by Lemma 4.6.7, an extension of $\mathbf{t}_H^{\Phi(\alpha)} = \mathbf{r}_{H_1}^{\Phi_1(\alpha)} \otimes_\Phi \mathbf{r}_{H_2}^{\Phi_2(\alpha)} \in \mathbb{F}^H$ of degree $|H_2| \cdot |H_1| + |H_1| \cdot |H_2| = 2|H|$ can be evaluated at β in time $\mathrm{polylog}(|H|) = \mathrm{poly}(\log|H_1|, \log|H_2|)$. □

Corollary 4.6.11. *Let $\Phi\colon H \to H_1 \times H_2$ be an efficient bivariate embedding. Let M be a (Φ,κ)-semisuccinct matrix, and let $d := |H_2| \cdot (|H_1| + \kappa(|H_1|))$. For every $\alpha \in \mathbb{F}$, a degree-d extension of $\mathbf{t}_H^{\Phi(\alpha)} M \in \mathbb{F}^H$ can be evaluated at any $\beta \in \mathbb{F}$ in time $\mathrm{poly}(\log|H_1|, |H_2|)$.*

Proof. The polynomials Φ_1 and Φ_2 can be evaluated at α in $\mathrm{polylog}(|H|)$ field operations, since Φ is efficient. We can write $M = A \otimes_\Phi B$ where $A \in \mathbb{F}^{H_1 \times H_1}$ is κ-succinct and $B \in \mathbb{F}^{H_2 \times H_2}$ is arbitrary, since M is (Φ,κ)-semisuccinct (see Definition 4.6.3). Since A is κ-succinct, we can evaluate a degree-$\kappa(|H_1|)$ extension of $\mathbf{r}_{H_1}^{\Phi_1(\alpha)} A$ at β in time $\mathrm{polylog}(|H|)$ (see Definition 4.6.2). We can evaluate a degree-$|H_2|$ extension of $\mathbf{r}_{H_2}^{\Phi_2(\alpha)} B$ at β in time $\mathrm{poly}(\log|H|, |H_2|)$ by direct interpolation. Thus, by Lemma 4.6.7, we can evaluate a degree-d extension of $\mathbf{t}_H^{\Phi(\alpha)} M = (\mathbf{r}_{H_1}^{\Phi_1(\alpha)} A) \otimes_\Phi (\mathbf{r}_{H_2}^{\Phi_2(\alpha)} B)$ in time $\mathrm{poly}(\log|H|, |H_2|)$. □

4.6.3 Proof of Lemma 4.6.4

We describe *succinct lincheck*, the RS oracle reduction that proves Lemma 4.6.4. Below we use as a subroutine the univariate sumcheck protocol (Protocol 3.4.3), which is an RS oracle reduction $(P_{\mathrm{SUM}}, V_{\mathrm{SUM}})$ for the relation $\mathcal{R}_{\mathrm{SUM}}$ of instance-witness pairs $(\mathbb{x}_{\mathrm{SUM}}, \mathbb{w}_{\mathrm{SUM}}) = \big((\mathbb{F}, L, H, \rho, \mu), f\big)$ such that $\mathbb{F}$ is a finite field, L is a subset of $\mathbb{F}$, H is a coset in $\mathbb{F}$, $\rho \in (0,1)$ is a rate parameter, μ is an element in $\mathbb{F}$, f is a codeword in $\mathrm{RS}[L,\rho]$, and $\sum_{a\in H} \hat{f}(a) = \mu$. In this (non-interactive) reduction, the prover sends a proof oracle π_Σ to the verifier, and the verifier outputs the rates $(\rho_0,\rho_1,\rho_2) = (\rho, \rho - |H|/|L|, (|H|-1)/|L|)$ and the virtual oracles $(\mathbf{\Pi}_0, \mathbf{\Pi}_1, \mathbf{\Pi}_2) = (f, \pi_\Sigma, \mathbf{\Pi}_\Sigma[f, \pi_\Sigma])$ for some $\mathbf{\Pi}_\Sigma$ (whose exact form depends on H and μ).

Protocol 4.6.12. Let $\mathbb{F}$ be a finite field, and $\Phi\colon H \to H_1 \times H_2$ an efficient bivariate embedding in $\mathbb{F}$ where H is a coset in $\mathbb{F}$; set $N := |H_1|$ and $n := |H_2|$. Succinct lincheck is a RS oracle reduction (P, V) that works for lincheck instances $\mathbb{x} = (\mathbb{F}, L, H, \rho, M)$ for which the matrix M has the form $\sum_{i=1}^{\ell} M_i \in \mathbb{F}^{H\times H}$ and each matrix $M_i \in \mathbb{F}^{H\times H}$ is (Φ,κ)-semisuccinct.

We describe the interaction between a prover P and verifier V that both receive an input a lincheck instance $\mathtt{x}$ as above, and where the prover P additionally receives a lincheck witness (f_1, f_2) (see Definition 3.5.1).

First, the verifier V draws a uniformly random $\alpha \in \mathbb{F}$ and sends it to the prover P. The element α defines polynomials $\hat{p}_\alpha^{(1)}(Y)$ and $\hat{p}_\alpha^{(2)}(Y)$ in $\mathbb{F}[Y]$ known to both prover and verifier:

- $\hat{p}_\alpha^{(1)}(Y)$, a degree-$2nN$ extension of the vector $\mathbf{t}_H^{\Phi(\alpha)} \in \mathbb{F}^H$ (see Definition 4.6.8). By Corollary 4.6.10, $\hat{p}_\alpha^{(1)}(Y)$ can be evaluated anywhere in $\mathrm{poly}(\log N, \log n)$ field operations.
- $\hat{p}_\alpha^{(2)}(Y)$, a degree-$(n \cdot (N + \kappa(N)))$ extension of the vector $\mathbf{t}_H^{\Phi(\alpha)} M \in \mathbb{F}^H$. Note that

$$\mathbf{t}_H^{\Phi(\alpha)} M = \mathbf{t}_H^{\Phi(\alpha)} \cdot \left(\sum_{i=1}^{\ell} M_i \right) = \sum_{i=1}^{\ell} \mathbf{t}_H^{\Phi(\alpha)} M_i \ .$$

  Each M_i is (Φ, κ)-semisuccinct so, by Corollary 4.6.11, a degree-$(n \cdot (N + \kappa(N)))$ extension of $\mathbf{t}_H^{\Phi(\alpha)} M_i$ can be evaluated anywhere in $\mathrm{poly}(\log N, n)$ field operations. By linearity, $\hat{p}_\alpha^{(2)}(Y)$, which is a degree-$(n \cdot (N + \kappa(N)))$ extension of $\mathbf{t}_H^{\Phi(\alpha)} M$, can be evaluated anywhere in $\mathrm{poly}(\ell, \log N, n)$ field operations.

Moreover, the element α and the witness codewords $f_1, f_2 \in \mathrm{RS}[L, \rho]$ jointly define the polynomial $\hat{q}_\alpha(Y)$ in $\mathbb{F}[Y]$ defined as follows:

$$\hat{q}_\alpha(Y) := \hat{p}_\alpha^{(1)}(Y) \hat{f}_1(Y) - \hat{p}_\alpha^{(2)}(Y) \hat{f}_2(Y) \ .$$

Observe that $\hat{q}_\alpha(Y)$ allegedly sums to zero on H, and has degree $\max\{2nN, n \cdot (N + \kappa(N))\}$, which is $n \cdot (N + \kappa(N))$ since $\kappa(N) \geq N$. In particular, $q_\alpha = \hat{q}_\alpha|_L$ (the restriction of the polynomial $\hat{q}_\alpha(Y)$ to the domain L) is a codeword in $\mathrm{RS}[L, \rho']$ for the rate parameter $\rho' := \rho + \frac{n \cdot (N + \kappa(N))}{|L|}$.

Next, the prover P and verifier V assemble the sumcheck instance $\mathtt{x}_{\mathrm{SUM}} = (\mathbb{F}, L, H, \rho', 0)$, and then run the univariate sumcheck protocol (an oracle reduction) with P playing the role of $P_{\mathrm{SUM}}^{q_\alpha}(\mathtt{x}_{\mathrm{SUM}})$ and V playing the role of $V_{\mathrm{SUM}}(\mathtt{x}_{\mathrm{SUM}})$. This results in $P_{\mathrm{SUM}}^{q_\alpha}(\mathtt{x}_{\mathrm{SUM}})$ sending a proof oracle π_Σ and then $V_{\mathrm{SUM}}(\mathtt{x}_{\mathrm{SUM}})$ outputting a list of instances and corresponding virtual oracles, as required of an oracle reduction.

Since $(P_{\mathrm{SUM}}, V_{\mathrm{SUM}})$ is an RS oracle reduction, we know that the instances output by V_{SUM} are rate parameters for the relation $\mathcal{R}_{\mathrm{RS}}$ over the domain L (see Definition 4.4.6). In the particular case at hand, V_{SUM} outputs the rate parameters $(\rho', \rho' - |H|/|L|, (|H|-1)/|L|)$ and corresponding virtual oracles $(q_\alpha, \pi_\Sigma, \mathbf{\Pi}_\Sigma[q_\alpha, \pi_\Sigma])$.

Finally, the verifier V outputs the rate parameters $(\rho, \rho, \rho_1, \rho_2) := (\rho, \rho, \rho' - |H|/|L|, (|H| - 1)/|L|)$ and the virtual oracles $(\mathbf{\Pi}_{f_1}, \mathbf{\Pi}_{f_2}, \mathbf{\Pi}_1, \mathbf{\Pi}_2)$ defined as follows:

$$\mathbf{\Pi}_{f_1}[f_1, f_2, \pi_\Sigma] := f_1 \quad \mathbf{\Pi}_{f_2}[f_1, f_2, \pi_\Sigma] := f_2 \quad \mathbf{\Pi}_1[f_1, f_2, \pi_\Sigma] := \pi_\Sigma \quad \mathbf{\Pi}_2[f_1, f_2, \pi_\Sigma] := \mathbf{\Pi}_\Sigma[q_\alpha, \pi_\Sigma]$$

Queries to q_α are simulated via queries to f_1 and f_2.

Note that since $(P_{\mathrm{SUM}}, V_{\mathrm{SUM}})$ is an RS oracle reduction then so is (P, V): each of the virtual oracles output by V are rational constraints, and every oracle sent by the prover appears as some virtual oracle.

Since $\mathbf{t}_H^\Phi \in \mathbb{F}[X]^H$ is a vector of linearly independent polynomials, the lincheck condition holds if and only if a certain polynomial equation in X holds:

$$\left\{\hat{f}_1(a) = \sum_{b\in H} M_{a,b}\cdot \hat{f}_2(b)\right\}_{a\in H} \quad\longleftrightarrow\quad \sum_{a\in H}\mathbf{t}_H^\Phi[a]\hat{f}_1(a) \equiv \sum_{a\in H}\mathbf{t}_H^\Phi[a]\sum_{b\in H} M_{a,b}\hat{f}_2(b) \ .$$

Rearranging the right-hand side of the polynomial equation yields:

$$\sum_{a\in H}\mathbf{t}_H^\Phi[a]\cdot\sum_{b\in H} M_{a,b}\hat{f}_2(b) \equiv \sum_{b\in H}\hat{f}_2(b)\cdot\sum_{a\in H}\mathbf{t}_H^\Phi[a]M_{a,b} \equiv \sum_{b\in H}\hat{f}_2(b)\cdot(\mathbf{t}_H^\Phi M)[b] \ .$$

For any choice of $\alpha \in \mathbb{F}$, we can evaluate each side of the polynomial equation:

$$\sum_{a\in H}\hat{p}_\alpha^{(1)}(a)\hat{f}_1(a) = \left(\sum_{a\in H}\mathbf{t}_H^\Phi[a]\hat{f}_1(a)\right)(\alpha) \ ,$$
$$\sum_{b\in H}\hat{p}_\alpha^{(2)}(b)\hat{f}_2(b) = \left(\sum_{b\in H}(\mathbf{t}_H^\Phi M)[b]\hat{f}_2(b)\right)(\alpha) \ .$$

These evaluations are equal if and only if $\sum_{a\in H}\hat{q}_\alpha(a) = 0$.

Completeness. Suppose that the lincheck condition holds. This implies that f_1 is a codeword in $\mathrm{RS}[L,\rho]$, and thus so is its corresponding virtual oracle, $\mathbf{\Pi}_{f_1}[f_1,f_2,\pi_\Sigma]$. Similarly for f_2 and $\mathbf{\Pi}_{f_2}[f_1,f_2,\pi_\Sigma]$. Moreover, for *every* $\alpha\in\mathbb{F}$ it holds that $\sum_{a\in H}\hat{q}_\alpha(a) = 0$, which means that $(\mathbb{x}_{\mathrm{SUM}}, q_\alpha)$ is a valid instance-witness pair for the sumcheck relation, and so completeness of the sumcheck protocol implies that $\mathbf{\Pi}_1[f_1,f_2,\pi_\Sigma] = \pi_\Sigma$ is a codeword in $\mathrm{RS}[L,\rho_1]$, and $\mathbf{\Pi}_2[f_1,f_2,\pi_\Sigma] = \mathbf{\Pi}_\Sigma[q_\alpha,\pi_\Sigma]$ is a codeword in $\mathrm{RS}[L,\rho_2]$.

Soundness. Suppose that the lincheck condition does not hold. If either $f_1 = \mathbf{\Pi}_{f_1}[f_1,f_2,\pi_\Sigma]$ or $f_2 = \mathbf{\Pi}_{f_2}[f_1,f_2,\pi_\Sigma]$ is not a codeword in $\mathrm{RS}[L,\rho]$, then we are done. So suppose instead that f_1, f_2 are codewords in $\mathrm{RS}[L,\rho]$, which means that $q_\alpha \in \mathrm{RS}[L,\rho']$. In this case there must exist $a\in H$ such that $\hat{f}_1(a) \neq \sum_{b\in H_0} M_{a,b}\cdot\hat{f}_2(b)$. With probability at least $1 - \frac{n\cdot(N+\kappa(N))}{|\mathbb{F}|}$, it holds that $\sum_{a\in H}\hat{q}_\alpha(a)\neq 0$. By soundness of the sumcheck protocol, either $\pi_\Sigma \notin \mathrm{RS}[L,\rho_1]$ or $\mathbf{\Pi}_\Sigma[q_\alpha,\pi_\Sigma]\notin\mathrm{RS}[L,\rho_2]$. This means that either $\mathbf{\Pi}_1[f_1,f_2,\pi_\Sigma]\notin\mathrm{RS}[L,\rho_1]$ or $\mathbf{\Pi}_2[f_1,f_2,\pi_\Sigma]\notin\mathrm{RS}[L,\rho_2]$, and again we are done. (The distance in the statement of Lemma 4.6.4 follows via an application of Lemma 4.4.7.)

Efficiency. The length of the reduction is the same as that of the sumcheck protocol, which is $|L|$. The locality is one more than that of the sumcheck protocol (since a query to q_α translates to a query to each of f_1 and f_2), for a total of 3. The running time of the verifier is dominated by the cost of constructing the virtual oracles $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$ each of which requires producing a circuit that answers a query to q_α by combining answers to queries to f_1 and f_2. This requires producing circuits for evaluating $\hat{p}_\alpha^{(1)}$ and $\hat{p}_\alpha^{(2)}$ on a point in L. Corollaries 4.6.10 and 4.6.11 give the claimed running time.

4.6.4 Extension to block-matrix lincheck

The lincheck relation in Definition 3.5.1 is a special case of a relation that we use in the proof of Lemma 4.7.2 (see Section 4.7.2), in order to obtain an oracle reduction for interactive

R1CS automata. We now describe the more general relation, and then explain how the ideas discussed so far directly extend to handle it.

The lincheck relation requires checking that $v_1 = Mv_2$ where $v_1, v_2 \in \mathbb{F}^H$ are encoded by Reed–Solomon codewords $f_1, f_2 \in \mathrm{RS}[L, \rho]$ respectively. Later on, we will want to check such conditions for vectors $v_1 := v_1^{(1)} \| \dots \| v_i^{(r)}$ and $v_2 := v_2^{(1)} \| \dots \| v_2^{(s)}$ such that each $v_1^{(i)}, v_2^{(j)} \in \mathbb{F}^H$ is *individually* encoded by a Reed–Solomon codeword $f_1^{(i)}, f_2^{(j)}$ respectively. We decompose M into several $H \times H$ block matrices $M^{(i,j)}$, so that $v_1 = Mv_2$ if and only if, for all $i \in [r]$, it holds that $v_1^{(i)} = \sum_{j=1}^{s} M^{(i,j)} v_2^{(j)}$. The block-matrix form of the lincheck relation is obtained by re-writing this condition in terms of codewords $f_1^{(i)}, f_2^{(j)}$.

Definition 4.6.13. *For $r, s \in \mathbb{N}$, the* **block-matrix lincheck relation** $\mathcal{R}_{\mathrm{LIN}}^{r,s}$ *consists of instance-witness pairs*

$$(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, H, \rho, \mathbf{M}), (\mathbf{f}_1, \mathbf{f}_2)\big)$$

where $\mathbb{F}$ is a finite field, L, H are subsets of $\mathbb{F}$, ρ is a rate parameter in $(0,1)$, $\mathbf{f}_1 = (f_1^{(1)}, \dots, f_1^{(r)})$ and $\mathbf{f}_2 = (f_2^{(1)}, \dots, f_2^{(s)})$ are lists of codewords in $\mathrm{RS}[L, \rho]$, $\mathbf{M} = (M^{(i,j)})_{i \in [r], j \in [s]}$ is a block matrix with each $M_{(i,j)} \in \mathbb{F}^{H \times H}$, and for all $i \in [r]$ and $a \in H$ it holds that $\hat{f}_1^{(i)}(a) = \sum_{j=1}^{s} \sum_{b \in H} M_{a,b}^{(i,j)} \hat{f}_2^{(j)}(b)$.

The succinctness condition that we now consider on the block matrix is that *each* block $M^{(i,j)}$ is a sum of semisuccinct matrices. We can then extend Protocol 4.6.12, in a straightforward way, to obtain an oracle reduction for the block-matrix lincheck relation $\mathcal{R}_{\mathrm{LIN}}^{r,s}$. Informally, the verifier's first message contains, in addition to $\alpha \in \mathbb{F}$, random elements $\beta_1, \dots, \beta_r \in \mathbb{F}$. We then consider the virtual oracle q induced by the polynomial

$$\hat{q}(Y) := \sum_{i=1}^{r} \beta_i \left(\hat{p}_\alpha^{(1)}(Y) \hat{f}_1^{(i)}(Y) - \sum_{j=1}^{s} \hat{p}_\alpha^{(i,j)}(Y) \hat{f}_2^{(i)}(Y) \right)$$

where $\hat{p}_\alpha^{(i,j)}$ is defined like $\hat{p}_\alpha^{(2)}$ but with respect to the matrix $M^{(i,j)} \in \mathbb{F}^{H \times H}$.

The foregoing ideas allow us to extend Lemma 4.6.4 to the block-matrix lincheck relation, and we obtain the following lemma, which we state without proof. The verifier uses $\mathrm{poly}(|\mathbb{x}|) = \mathrm{poly}(\ell, r, s, n, \log N)$ field operations, which is *polylogarithmic* in the size of the succinct part of the matrix.

Lemma 4.6.14. *Let $\mathbb{F}$ be a finite field with an efficient bivariate embedding $\Phi \colon H \to H_1 \times H_2$ where H is a coset in $\mathbb{F}$. Suppose that $\mathbf{M} = (M^{(i,j)})_{i \in [r], j \in [s]}$ is a block matrix where each block has the form $M^{(i,j)} = \sum_{\iota=1}^{\ell} M_\iota^{(i,j)}$ and each $M_\iota^{(i,j)} \in \mathbb{F}^{H \times H}$ is (Φ, κ)-semisuccinct.*

Setting $N := |H_1|$, $n := |H_2|$, and $d := n \cdot (N + \kappa(N))$, there is a 1-round RS oracle reduction of proximity for $\mathcal{R}_{\mathrm{LIN}}^{r,s}$ for instances of the form $\mathbb{x} = (\mathbb{F}, L, H, \rho, \mathbf{M})$ with the following parameters:

length	$\lvert L \rvert$	*soundness error*	$(d+1)/\lvert\mathbb{F}\rvert$	*prover time*	$\lvert L \rvert \log \lvert L \rvert + \lVert \mathbf{M} \rVert$
locality	2	*distance*	$\frac{1}{2}(1 - \rho - \frac{d}{\lvert L \rvert})$	*verifier time*	$\mathrm{poly}(\lvert \mathbb{x} \rvert)$

(Above $\lVert \mathbf{M} \rVert$ denotes the total number of non-zero entries across all blocks of the matrix $\mathbf{M} = (M^{(i,j)})_{i \in [r], j \in [s]}$.)

4.7 Probabilistic checking of interactive automata

We define the *R1CS automata* relation, which considers checking algebraic computation that has no external memory, and show how to reduce it to the lincheck relation (see Definition 4.6.13), via an oracle reduction. Combining this reduction with results in Section 4.6, we obtain an oracle reduction from R1CS automata to testing proximity to the Reed–Solomon code (see Lemma 4.7.2 below). This oracle reduction is later used to obtain an oracle reduction for a relation about *R1CS machines* (see Section 4.8), which have external memory.

Informally, we define computation on R1CS automata as follows. Let $\mathbb{F}$ be a finite field, $T \in \mathbb{N}$ be a computation *time*, and $k \in \mathbb{N}$ a computation *width*. We consider execution traces $f\colon [T] \to \mathbb{F}^k$ that represent T-time computations in which each state $f(t)$ of the computation is a vector of k elements in $\mathbb{F}$. An *R1CS automaton* is specified by matrices $A, B, C \in \mathbb{F}^{k \times 2k}$ that define R1CS time constraints, and a set of boundary constraints $\mathcal{B} \subseteq [T] \times [k] \times \mathbb{F}$. An execution trace $f\colon [T] \to \mathbb{F}^k$ is accepted by the automaton if:

- f satisfies the R1CS *time constraints*, namely, for every $t \in [T-1]$, letting $f(t, t+1)$ be the concatenation of the consecutive states $f(t) \in \mathbb{F}^k$ and $f(t+1) \in \mathbb{F}^k$, it holds that $Af(t, t+1) \circ Bf(t, t+1) = Cf(t, t+1)$;
- f satisfies the *boundary constraints*, namely, for every $(t, j, \alpha) \in \mathcal{B}$ it holds that $f(t)_j = \alpha$.

Intuitively, time constraints enforce that each state in the execution trace is consistent with the prior one. Boundary constraints enforce that given locations in the execution trace have given values, for example, they could ensure that the computation started at a certain initial state and halted at a certain final state.

Interactive automata. We shall in fact consider a more general notion of computation that allows interaction with a prover, which we call *interactive R1CS automata*. This notion enables an efficient oracle reduction from *R1CS machines*, as described in Section 4.8.

Informally, in *interactive* R1CS automata, instead of considering execution traces $f\colon [T] \to \mathbb{F}^k$, we consider prover strategies P_{w} that output an execution *sub*-trace $w_i\colon [T] \to \mathbb{F}^s$ in each round of a public-coin protocol of r rounds. By juxtaposing the sub-traces $w_1, \ldots, w_r$ one obtains a trace $f\colon [T] \to \mathbb{F}^{rs}$, which is then accepted if it satisfies time constraints and boundary constraints similarly as above. Crucially, the matrices defining time constraints can depend on the verifier's randomness in the public-coin protocol.

Below we define a universal relation $\mathcal{R}^{r,\epsilon}_{\mathrm{R1A}}$ that captures computations on r-round interactive R1CS automata with soundness error ϵ. We denote the computation time by T, the length of a verifier message by ℓ, the width of a sub-trace by s, and the width of a trace by $k := rs$. Given a trace $f\colon [T] \to \mathbb{F}^k$, we denote by $f(i, j) \in \mathbb{F}^{2k}$ the concatenation of the two states $f(i) \in \mathbb{F}^k$ and $f(j) \in \mathbb{F}^k$.

Definition 4.7.1. *The promise relation $\mathcal{R}^{r,\epsilon}_{\mathrm{R1A}} = (\mathcal{R}_{\mathsf{Yes}}, \mathcal{L}_{\mathsf{No}})$ of bounded accepting computation problems on* **interactive R1CS automata** *is defined as follows. An instance $\mathbb{x} = (\mathbb{F}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathcal{B}, \ell)$ consists of a finite field $\mathbb{F}$, functions $\mathbf{A}, \mathbf{B}, \mathbf{C}\colon \mathbb{F}^{r\ell} \to \mathbb{F}^{s \times 2rs}$ defining time constraints, and boundary constraints $\mathcal{B} \subseteq [T] \times [rs] \times \mathbb{F}$. A witness P_{w} is a prover strategy for the following game defined by $\mathbb{x}$. Let $V_{\mathbb{x}}$ be the interactive Turing machine that interacts with P_{w}*

for r rounds, where in the i-th round $V_{\mathbb{x}}$ sends a random $a_i \in \mathbb{F}^\ell$, and $P_{\mathbb{w}}$ replies with a message $w_i \colon [T] \to \mathbb{F}^s$. At the end of the interaction, letting $a := (a_1, \ldots, a_r) \in \mathbb{F}^{r\ell}$ and $f \colon [T] \to \mathbb{F}^{rs}$ be defined as $f(t) := w_1(t) \| \ldots \| w_r(t)$, $V_{\mathbb{x}}$ accepts if the following holds.

- Time constraints: $\forall\, t \in [T-1],\ \mathbf{A}(a) f(t, t+1) \circ \mathbf{B}(a) f(t, t+1) = \mathbf{C}(a) f(t, t+1)\,.$
- Boundary constraints: $\forall\, (t, j, \alpha) \in \mathcal{B},\ f(t)_j = \alpha\,.$

A pair $(\mathbb{x}, P_{\mathbb{w}})$ is in $\mathcal{R}_{\mathsf{Yes}}$ if $(P_{\mathbb{w}}, V_{\mathbb{x}})$ accepts with probability 1. On the other hand, an instance $\mathbb{x}$ is in $\mathcal{L}_{\mathsf{No}}$ if for every prover strategy $\tilde{P}$ it holds that $(\tilde{P}, V_{\mathbb{x}})$ accepts with probability at most ϵ.

In this section we obtain an oracle reduction, with linear length and locality $r+1$, from interactive R1CS automata to testing proximity to the Reed–Solomon code. The verifier uses $\mathrm{poly}(|\mathbb{x}|) = \mathrm{poly}(r, s, |\mathcal{B}|, \log T)$ field operations, which is *exponentially* faster than the computation time T, since the dependence on the T is polylogarithmic instead of polynomial.

Lemma 4.7.2. *Let $\mathbb{F}$ be a finite field, H a coset in $\mathbb{F}$, and L a subset of $\mathbb{F}$ with $L \cap H = \emptyset$. Let $\mathcal{T} \colon [T] \times [s] \to H$ be a trace embedding in $\mathbb{F}$. There is an RS oracle reduction over domain L (Protocol 4.7.7) for instances $\mathbb{x}$ of $\mathcal{R}^{r,\epsilon}_{\mathrm{R1A}}$ over $\mathbb{F}$ and of computation time T and width $k = rs$. The reduction has $r+1$ rounds and the following parameters:*

length	$(r+4)\lvert L\rvert$	*soundness error*	$\epsilon + (3sT+1)/\lvert\mathbb{F}\rvert$	*prover time*	$\lvert L\rvert \cdot (\log \lvert L\rvert + \lvert\mathbb{x}\rvert)$
locality	$r+1$	*distance*	$\frac{1}{2}(1 - 4sT/\lvert L\rvert)$	*verifier time*	$\mathrm{poly}(\lvert\mathbb{x}\rvert)$

The rest of this section is dedicated to proving Lemma 4.7.2. Our high-level approach is to first identify a family of *semisuccinct* matrices (see Definition 4.6.3) that can express computation via R1CS automata. We call this family *staircase matrices* and, in Section 4.7.1, prove that they are indeed semisuccinct. Then, in Section 4.7.2 we prove Lemma 4.7.2 by invoking the succinct lincheck protocol in Section 4.6 (which requires semisuccinct matrices) on carefully chosen staircase matrices, derived from an interactive automaton.

4.7.1 Staircase matrices

We introduce the notion of *staircase matrices* and prove that they are the sum of two semisuccinct matrices. This requires establishing simple algebraic properties of the identity matrix and a related matrix.

First recall from Definition 4.6.2 that a matrix $A \in \mathbb{F}^{S \times S}$ is *κ-succinct* if, for every $d \geq |S| - 1$ and $v \in \mathbb{F}^S$, a degree-$\kappa(d)$ extension of vA can be evaluated anywhere in $\mathbb{F}$ in time $\mathrm{polylog}(|S|)$ whenever a degree-d extension of v can be evaluated anywhere in $\mathbb{F}$ in time $\mathrm{polylog}(|S|)$. The *identity matrix* on a subset S of $\mathbb{F}$ (the matrix $I \in \mathbb{F}^{S \times S}$ with $I(\alpha, \alpha) = 1$ for all $\alpha \in S$ and 0 elsewhere) is trivially κ-succinct for $\kappa(d) := d$: if a degree-d extension of v can be evaluated anywhere in $\mathbb{F}$ in time T, then so can a degree-d extension of vI.

We now define the *shifted identity matrix*, for a given successor ordering, and prove that it is κ-succinct, where κ depends on algebraic properties of the successor ordering on the subset S that we consider. Recall from Definition 4.5.13 that a *successor ordering* on

$S \subseteq \mathbb{F}$ is a pair $\mathcal{O} = (\mathbf{1}_S, N)$ where $\mathbf{1}_S \in S$ and N is a piecewise polynomial on S of degree 1 such that $S = \{\alpha_1, \ldots, \alpha_{|S|}\}$, where $\alpha_1 := \mathbf{1}_S$ and $\alpha_{i+1} := N(\alpha_i)$ inductively for every $i \in \{1, \ldots, |S| - 1\}$.

Definition 4.7.3. *Let $\mathbb{F}$ be a field, S a subset of $\mathbb{F}$, and $\mathcal{O} = (\mathbf{1}_S, N)$ a successor ordering on S. The* **shifted identity matrix** *$I^{\rightarrow}_{\mathcal{O}} \in \mathbb{F}^{S \times S}$ has $I^{\rightarrow}_{\mathcal{O}}(\alpha, N(\alpha)) = 1$ for all $\alpha \in S$ such that $\gamma_{\mathcal{O}}(\alpha) < |S|$, and 0 elsewhere. Under the ordering $\gamma_{\mathcal{O}}$, we can view $I^{\rightarrow}_{\mathcal{O}}$ as the following matrix:*

$$I^{\rightarrow}_{\mathcal{O}} = \begin{pmatrix} 0 & 1 & & \\ & 0 & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{pmatrix} .$$

Lemma 4.7.4. *Let S be a subset of $\mathbb{F}$ whose vanishing polynomial can be computed in $\mathrm{polylog}(|S|)$ field operations and $\mathcal{O} = (\mathbf{1}_S, N)$ be an efficient successor ordering on S (see Definition 4.5.13). Then $I^{\rightarrow}_{\mathcal{O}}$ is κ-succinct for $\kappa(d) := |S| + d$.*

Proof. Let $r \in \mathbb{F}^S$ and $d \geq |S|$ be such that a degree-d extension $\hat{r} \in \mathbb{F}[X]$ of r can be evaluated in T field operations. We prove that a degree-$(|S| + d)$ extension $\hat{r}^{\rightarrow}_{\mathcal{O}} \in \mathbb{F}[X]$ of $r^{\rightarrow}_{\mathcal{O}} := r I^{\rightarrow}_{\mathcal{O}} \in \mathbb{F}^S$ can be evaluated in $O(T + \mathrm{polylog}(|S|))$ field operations.

Observe that $r^{\rightarrow}_{\mathcal{O}} = (0, r_1, \ldots, r_{|S|-1})$ is the right shift of r, so that, for every $\alpha \in S$, we have

$$r^{\rightarrow}_{\mathcal{O}}(\alpha) = r(N(\alpha)) - r(N(\mathbf{1}_S)) \cdot \mathbb{I}[\alpha = \mathbf{1}_S]$$

where $\mathbb{I}[\alpha = \mathbf{1}_S]$ is the indicator function for $\mathbf{1}_S$ on S. Let $(\mathcal{S}, \mathcal{F}) = \big((S_1, \ldots, S_\ell), (f_1, \ldots, f_\ell)\big)$ be a piecewise polynomial of piecewise degree 1 that computes N (see Definition 4.5.11), and let s_i be the unique extension of degree less than $|S|$ of the indicator for S_i in S. We have that

$$r(N(\alpha)) = r\left(\sum_{i=1}^{\ell} s_i(\alpha) f_i(\alpha)\right) = \sum_{i=1}^{\ell} s_i(\alpha) r(f_i(\alpha)) \ .$$

We deduce that

$$\hat{r}^{\rightarrow}_{\mathcal{O}}(X) := \left(\sum_{i=1}^{\ell} s_i(X) \cdot \hat{r}(f_i(X))\right) - \left(\sum_{i=1}^{\ell} s_i(\mathbf{1}_S) \cdot \hat{r}(f_i(\mathbf{1}_S))\right) \cdot L_{S,\mathbf{1}_S}(X)$$

is a degree-$(|S| + d)$ extension of $r^{\rightarrow}_{\mathcal{O}}$. Note that $L_{S,\mathbf{1}_S}$ can be evaluated in $\mathrm{polylog}(|S|)$ operations, since the vanishing polynomial of S can be evaluated in $\mathrm{polylog}(|S|)$ operations. (See Section 2.1.1). Hence, taking also into account that N is efficient, we conclude that $\hat{r}^{\rightarrow}_{\mathcal{O}}$ can be evaluated in $O(T + \mathrm{polylog}(|S|))$ operations. □

The staircase matrix of two matrices M and M' is the block matrix whose diagonal consists of blocks of M and its superdiagonal consists of blocks of M'. Algebraically, we capture this by considering the matrix that consists of the sum of two terms: (1) the tensor product of the identity matrix with M; and (2) the tensor product of the shifted identity matrix with M'. We

formally define this notion, and then use Lemma 4.7.4 to deduce that each of these two terms is semisuccinct. Recall from Definition 4.6.3 that a matrix $M \in \mathbb{F}^{H \times H}$ is (Φ, κ)-*semisuccinct*, where $\Phi \colon H \to H_1 \times H_2$ is a bivariate embedding and $\kappa \colon \mathbb{N} \to \mathbb{N}$, if M can be written as $A \otimes_\Phi B$ for κ-succinct $A \in \mathbb{F}^{H_1 \times H_1}$ and arbitrary $B \in \mathbb{F}^{H_2 \times H_2}$.

Definition 4.7.5. *Let $\mathbb{F}$ be a finite field, $\Phi \colon H \to H_1 \times H_2$ a bivariate embedding in $\mathbb{F}$, and $\mathcal{O}$ a successor ordering on H_1. The* **staircase matrix** *of two matrices $M, M' \in \mathbb{F}^{H_2 \times H_2}$ is the matrix in $\mathbb{F}^{H \times H}$ defined as*

$$S_{\Phi,\mathcal{O}}(M, M') := I \otimes_\Phi M + I^{\rightarrow}_{\mathcal{O}} \otimes_\Phi M'$$

where I and $I^{\rightarrow}_{\mathcal{O}}$ are the identity and shifted identity matrices in $\mathbb{F}^{H_1 \times H_1}$, respectively. Under the appropriate ordering on H, we can write $S_{\Phi,\mathcal{O}}(M, M')$ as the block matrix:

$$S_{\Phi,\mathcal{O}}(M, M') = \begin{pmatrix} M & M' & & & & \\ & M & M' & & & \\ & & M & M' & & \\ & & & \ddots & \ddots & \\ & & & & M & M' \\ & & & & & M \end{pmatrix}.$$

Corollary 4.7.6. *Let $\mathbb{F}$ be a finite field, $\Phi \colon H \to H_1 \times H_2$ an efficient bivariate embedding in $\mathbb{F}$, and $\mathcal{O} = (\mathbf{1}_{H_1}, N)$ an efficient successor ordering on H_1. Then each of the terms in a staircase matrix $S_{\Phi,\mathcal{O}}(M, M') = I \otimes_\Phi M + I^{\rightarrow}_{\mathcal{O}} \otimes_\Phi M'$ is (Φ, κ)-semisuccinct for $\kappa(d) := |H_1| + d$.*

4.7.2 Proof of Lemma 4.7.2

We show an oracle reduction, with linear length and locality $r + 1$, from the interactive R1CS automata relation $\mathcal{R}^{r,\epsilon}_{\mathrm{R1A}}$ to testing proximity to the Reed–Solomon code. Let $\mathbb{x} = (\mathbb{F}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathcal{B}, \ell)$ be an instance of the relation $\mathcal{R}^{r,\epsilon}_{\mathrm{R1A}}$, and let $P_{\mathbb{w}}$ be a candidate witness for $\mathbb{x}$. Recall that the witness $P_{\mathbb{w}}$ is a prover strategy for the r-round public-coin game defined by the verifier $V_{\mathbb{x}}$ determined by $\mathbb{x}$.

In round i of this game, $V_{\mathbb{x}}$ sends a uniformly random $a_i \in \mathbb{F}^\ell$, and $P_{\mathbb{w}}$ replies with a message $w_i \colon [T] \to \mathbb{F}^s$. Letting $a := (a_1, \ldots, a_r) \in \mathbb{F}^{r\ell}$ and $f \colon [T] \to \mathbb{F}^{rs}$ be defined as $f(t) := w_1(t) \| \ldots \| w_r(t)$, to check membership in the relation $\mathcal{R}^{r,\epsilon}_{\mathrm{R1A}}$ it suffices to verify the following.

(i) Time constraints: $\forall\, t \in [T-1]\,,\ \mathbf{A}(a)f(t, t+1) \circ \mathbf{B}(a)f(t, t+1) = \mathbf{C}(a)f(t, t+1)$.
(ii) Boundary constraints: $\forall\, (t, j, \alpha) \in \mathcal{B}\,,\ f(t)_j = \alpha$.

Our task now is to specify the prover P and the verifier V of a suitable Reed–Solomon oracle reduction. Informally, the prover P and verifier V engage in an "encoded" version of the game defined by $\mathbb{x}$. Then, the verifier uses the prover's encoded messages to reduce checking the time constraints and the boundary constraints to membership in the Reed–Solomon code.

The role of *staircase matrices* (defined in Section 4.7.1) for these checks can be explained as follows. Let $\mathbf{A}(a)_1$ be the first rs columns of $\mathbf{A}(a)$, and $\mathbf{A}(a)_2$ the remaining rs columns;

similarly for $\mathbf{B}(a)$ and $\mathbf{C}(a)$. Viewing f as a vector $f(1)\| \cdots \| f(T) \in \mathbb{F}^{kT}$, we left-multiply the vector f by staircase matrices, obtaining:

$$\begin{aligned} f_{\mathbf{A}} &:= S(\mathbf{A}(a)_1, \mathbf{A}(a)_2) \cdot f = \mathbf{A}(a) f(1,2) \| \cdots \| \mathbf{A}(a) f(T-1,T) \| \mathbf{A}(a)_1 f(T) \in \mathbb{F}^{sT} \ , \\ f_{\mathbf{B}} &:= S(\mathbf{B}(a)_1, \mathbf{B}(a)_2) \cdot f = \mathbf{B}(a) f(1,2) \| \cdots \| \mathbf{B}(a) f(T-1,T) \| \mathbf{B}(a)_1 f(T) \in \mathbb{F}^{sT} \ , \\ f_{\mathbf{C}} &:= S(\mathbf{C}(a)_1, \mathbf{C}(a)_2) \cdot f = \mathbf{C}(a) f(1,2) \| \cdots \| \mathbf{C}(a) f(T-1,T) \| \mathbf{C}(a)_1 f(T) \in \mathbb{F}^{sT} \ . \end{aligned}$$

(For simplicity, we suppress the bivariate embedding and successor ordering used to define a staircase matrix.)

The time constraints are equivalent to checking that $f_{\mathbf{A}}, f_{\mathbf{B}}, f_{\mathbf{C}}$ are consistent with f, which we can do via our new succinct lincheck protocol from Section 4.6, and also checking that $f_{\mathbf{A}} \circ f_{\mathbf{B}}$ and $f_{\mathbf{C}}$ agree on their first $s \cdot (T-1)$ entries, which we can do via other (standard) probabilistic checking tools.

We now provide a formal description of the reduction and then discuss its properties.

Protocol 4.7.7. Let $\mathbb{F}$ be a finite field, H a coset in $\mathbb{F}$, and L a subset of $\mathbb{F}$ with $L \cap H = \emptyset$. Let $\mathcal{T} = (\Phi \colon H \to H_1 \times H_2, \mathcal{O}, \gamma)$ be a trace embedding in $\mathbb{F}$ with $T = |H_1|$ and $s = |H_2|$ (see Definition 4.5.1). We show a Reed–Solomon oracle reduction over domain L, which works on instances $\mathbb{x} = (\mathbb{F}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathcal{B}, \ell)$ for the relation $\mathcal{R}^{r,\epsilon}_{\mathrm{R1A}}$ (see Definition 4.7.1) that have computation time T and width $k = rs$. The oracle reduction is specified by the prover P and verifier V described below. Recall that P and V receive the instance $\mathbb{x}$ as input, while P additionally receives a witness $P_{\mathbb{w}}$ for $\mathbb{x}$.

1. **Interaction.** The prover P and verifier V engage in an "encoded" version of the r-round game induced by $\mathbb{x}$. In round i, first V behaves exactly as $V_{\mathbb{x}}$ by sending random elements $a_i \in \mathbb{F}^\ell$; then P obtains a message $w_i \colon [T] \to \mathbb{F}^s$ from $P_{\mathbb{w}}$ and, instead of sending w_i, sends its encoding $\pi_{w_i} := \hat{\pi}_{w_i}|_L$, where $\hat{\pi}_{w_i}$ is the unique polynomial of degree less than sT such that
$$\forall\, t \in [T]\,,\ \forall\, j \in [s] \quad \hat{\pi}_{w_i}\big(\mathcal{T}(t,j)\big) = (w_i(t))[j] \ .$$
For each $i \in [r]$, the verifier V outputs the rate parameter $\rho_{w_i} := sT/|L|$ and virtual oracle $\mathbf{\Pi}_{w_i} := \pi_{w_i}$.

2. **Proof oracles.** The prover P uses the verifier randomness $a = (a_1, \ldots, a_r) \in \mathbb{F}^{r\ell}$ to compute the matrices $\mathbf{A}(a), \mathbf{B}(a), \mathbf{C}(a) \in \mathbb{F}^{s \times 2rs}$. We view these as $2r$ block matrices with blocks of size $s \times s$:
$$\begin{aligned} \mathbf{A}(a) &= \begin{pmatrix} A_1^{(1)} & \cdots & A_1^{(r)} & A_2^{(1)} & \cdots & A_2^{(r)} \end{pmatrix} \ , \\ \mathbf{B}(a) &= \begin{pmatrix} B_1^{(1)} & \cdots & B_1^{(r)} & B_2^{(1)} & \cdots & B_2^{(r)} \end{pmatrix} \ , \\ \mathbf{C}(a) &= \begin{pmatrix} C_1^{(1)} & \cdots & C_1^{(r)} & C_2^{(1)} & \cdots & C_2^{(r)} \end{pmatrix} \ . \end{aligned}$$

Next, P computes the unique polynomials $\hat{\pi}_{\mathbf{A}}, \hat{\pi}_{\mathbf{B}}, \hat{\pi}_{\mathbf{C}}$ of degree less than sT such that

$$\begin{aligned}
\forall\, h \in H \qquad \hat{\pi}_{\mathbf{A}}(h) &= \Big(\sum_{i=1}^{r} S_{\Phi,\mathcal{O}}(A_1^{(i)}, A_2^{(i)}) \cdot \hat{\pi}_{w_i}|_H \Big)[h] \ , \\
\forall\, h \in H \qquad \hat{\pi}_{\mathbf{B}}(h) &= \Big(\sum_{i=1}^{r} S_{\Phi,\mathcal{O}}(B_1^{(i)}, B_2^{(i)}) \cdot \hat{\pi}_{w_i}|_H \Big)[h] \ , \\
\forall\, h \in H \qquad \hat{\pi}_{\mathbf{C}}(h) &= \Big(\sum_{i=1}^{r} S_{\Phi,\mathcal{O}}(C_1^{(i)}, C_2^{(i)}) \cdot \hat{\pi}_{w_i}|_H \Big)[h] \ .
\end{aligned}$$

Finally, P sends to V the codewords in $\mathrm{RS}[L, sT/|L|]$ obtained by restricting the above polynomials to L:

$$\pi_{\mathbf{A}} := \hat{\pi}_{\mathbf{A}}|_L \quad \pi_{\mathbf{B}} := \hat{\pi}_{\mathbf{B}}|_L \quad \pi_{\mathbf{C}} := \hat{\pi}_{\mathbf{C}}|_L \ .$$

The verifier V outputs rate parameters $(\rho_{\mathbf{A}}, \rho_{\mathbf{B}}, \rho_{\mathbf{C}})$ and virtual oracles $(\mathbf{\Pi}_{\mathbf{A}}, \mathbf{\Pi}_{\mathbf{B}}, \mathbf{\Pi}_{\mathbf{C}})$ defined as:

$$\begin{aligned}
\rho_{\mathbf{A}} &:= sT/|L| \qquad & \mathbf{\Pi}_{\mathbf{A}} &:= \pi_{\mathbf{A}} \ , \\
\rho_{\mathbf{B}} &:= sT/|L| \qquad & \mathbf{\Pi}_{\mathbf{B}} &:= \pi_{\mathbf{B}} \ , \\
\rho_{\mathbf{C}} &:= sT/|L| \qquad & \mathbf{\Pi}_{\mathbf{C}} &:= \pi_{\mathbf{B}} \ .
\end{aligned}$$

3. **Succinct lincheck.** The prover P and verifier V invoke the block-matrix succinct lincheck of Lemma 4.6.14 on the instance $\mathbb{x}_{\mathrm{LIN}} := (\mathbb{F}, L, H, \rho, \mathbf{M})$ and witness $\mathbb{w}_{\mathrm{LIN}} := ((\pi_{\mathbf{A}}, \pi_{\mathbf{B}}, \pi_{\mathbf{C}}), (\pi_{w_1}, \ldots, \pi_{w_r}))$, where

$$\mathbf{M} := \begin{pmatrix} S_{\Phi,\mathcal{O}}(A_1^{(1)}, A_2^{(1)}) & \cdots & S_{\Phi,\mathcal{O}}(A_1^{(r)}, A_2^{(r)}) \\ S_{\Phi,\mathcal{O}}(B_1^{(1)}, B_2^{(1)}) & \cdots & S_{\Phi,\mathcal{O}}(B_1^{(r)}, B_2^{(r)}) \\ S_{\Phi,\mathcal{O}}(C_1^{(1)}, C_2^{(1)}) & \cdots & S_{\Phi,\mathcal{O}}(C_1^{(r)}, C_2^{(r)}) \end{pmatrix} \ .$$

The verifier V outputs the rate parameters and virtual oracles output by the verifier of this RS oracle reduction. The maximum rate across these is $\rho + d/|L|$, where $\rho := sT/|L|$ and $d := s \cdot (T + \kappa(T)) = 3sT$, since $\kappa(T) = |H_1| + T = 2T$ by Corollary 4.7.6.

4. **Rowcheck.** Define the set $H_{\mathrm{ROW}} := H \setminus \{\mathcal{T}(T,1), \ldots, \mathcal{T}(T,s)\}$. The verifier V outputs the rate parameter ρ_{ROW} and virtual oracle $\mathbf{\Pi}_{\mathrm{ROW}}$ defined as:

$$\rho_{\mathrm{ROW}} := \frac{2sT - |H_{\mathrm{ROW}}|}{|L|} \quad \text{and} \quad \mathbf{\Pi}_{\mathrm{ROW}}(\alpha) := \frac{\pi_{\mathbf{A}}(\alpha) \cdot \pi_{\mathbf{B}}(\alpha) - \pi_{\mathbf{C}}(\alpha)}{\mathbb{Z}_{H_{\mathrm{ROW}}}(\alpha)} \ ,$$

where $\mathbb{Z}_{H_{\mathrm{ROW}}}(X) := \mathbb{Z}_{H_1}(\Phi_1(X))/\mathbb{Z}_{\{\mathcal{T}(T,1),\ldots,\mathcal{T}(T,s)\}}(X)$ is (a multiple of) the vanishing polynomial of H_{ROW} because $\mathbb{Z}_{H_1}(\Phi_1(X))$ is (a multiple of) the vanishing polynomial of H. Observe that $\mathbb{Z}_{H_{\mathrm{ROW}}}$ can be evaluated in $\mathrm{poly}(\log T, s)$ field operations because: (1) the definition of a trace embedding (Definition 4.5.1) requires that $\mathbb{Z}_{H_1}$ and Φ_1 can each be evaluated in $\mathrm{polylog}(|H|) = \mathrm{polylog}(Ts)$ field operations; (2) the denominator can be

evaluated in $\mathrm{poly}(s)$ field operations. (We also know that H is a coset in $\mathbb{F}$, a condition inherited from univariate sumcheck, which means that the vanishing polynomial of H can directly be evaluated in $\mathrm{polylog}(|H|)$ field operations. The above reasoning assumes less.)

The above is an RS oracle reduction of proximity over domain L for the *rowcheck relation* $\mathcal{R}_{\mathrm{ROW}}$, consisting of instance-witness pairs $(\mathbb{x}_{\mathrm{ROW}}, \mathbb{w}_{\mathrm{ROW}})$, where $\mathbb{x}_{\mathrm{ROW}} = (\mathbb{F}, L, H_{\mathrm{ROW}}, \rho_{\mathrm{ROW}})$ and $\mathbb{w}_{\mathrm{ROW}} = (\pi_1, \pi_2, \pi_3)$, such that $\pi_1, \pi_2, \pi_3 \in \mathrm{RS}\,[L, \rho]$ and, for every $\alpha \in H_{\mathrm{ROW}}$, $\hat{\pi}_1(\alpha) \cdot \hat{\pi}_2(\alpha) - \hat{\pi}_3(\alpha) = 0$.

5. **Enforce boundary constraints.** The verifier V partitions the boundary constraints $\mathcal{B}$ into $(\mathcal{B}_1, \ldots, \mathcal{B}_r)$ so that the constraints in $\mathcal{B}_i$ apply to the message $w_i \colon [T] \to \mathbb{F}^s$. Namely, for each $i \in [r]$, V defines

$$\mathcal{B}_i := \left\{ (t, j', \alpha) \;\middle|\; \exists\, j' \in \{1, \ldots, s\} \text{ s.t. } (t, j' + s \cdot (i-1), \alpha) \in \mathcal{B} \right\} .$$

Let $E_i := \{\mathcal{T}(t, j') : (t, j', \alpha) \in \mathcal{B}_i\}$ be the set of locations in H contained in $\mathcal{B}_i$. Let B_i be the polynomial of degree less than $|E_i|$ such that $B_i(\mathcal{T}(t, j')) = \alpha$ for every $(t, j', \alpha) \in \mathcal{B}_i$. The verifier outputs rate parameters $(\rho_{\mathcal{B}_1}, \ldots, \rho_{\mathcal{B}_r})$ and virtual oracles $(\mathbf{\Pi}_{\mathcal{B}_1}, \ldots, \mathbf{\Pi}_{\mathcal{B}_r})$ where each rate and virtual oracle is defined as follows:

$$\rho_{\mathcal{B}_i} := \frac{sT - |E_i|}{|L|} \quad \text{and} \quad \mathbf{\Pi}_{\mathcal{B}_i}(\alpha) := \frac{\pi_{w_i}(\alpha) - B_i(\alpha)}{\mathbb{Z}_{E_i}(\alpha)} .$$

We conclude the proof of Lemma 4.7.2 by showing its completeness, soundness, and efficiency.

Completeness. Suppose that $(\mathbb{x}, P_{\mathbb{w}}) \in \mathcal{R}_{\mathsf{Yes}}$, and consider the honest prover strategy P described above. We argue that every pair $(\rho, \mathbf{\Pi})$ output by the verifier V belongs to the Reed–Solomon relation $\mathcal{R}_{\mathrm{RS}}$ (see Definition 4.4.6). We separately consider each step in the reduction. In Item 1, for every $i \in [r]$, the virtual oracle $\mathbf{\Pi}_{w_i} = \pi_{w_i}$ indeed has rate $\rho_{w_i} = sT/|L|$. In Item 2, the virtual oracles $(\mathbf{\Pi}_{\mathbf{A}}, \mathbf{\Pi}_{\mathbf{B}}, \mathbf{\Pi}_{\mathbf{C}})$ indeed have rates $(\rho_{\mathbf{A}}, \rho_{\mathbf{B}}, \rho_{\mathbf{C}}) = (sT/|L|, sT/|L|, sT/|L|)$. In Item 3, the constructed instance-witness pair $(\mathbb{x}_{\mathrm{LIN}}, \mathbb{w}_{\mathrm{LIN}})$ satisfies the lincheck relation, and thus we rely on the completeness of the succinct lincheck protocol. In Item 4 and in Item 5, the constructed polynomials in the numerator are divisible by the denominator if and only if the rowcheck condition and boundary conditions hold respectively.

Soundness. Suppose that $\mathbb{x} \in \mathcal{L}_{\mathsf{No}}$. Suppose first that the oracles sent by the prover in Item 1 and Item 2 belong to the prescribed code:

$$\tilde{\pi}_{w_1} \in \mathrm{RS}\,[L, \rho_1], \ldots, \tilde{\pi}_{w_r} \in \mathrm{RS}\,[L, \rho_r]\,,\ \tilde{\pi}_{\mathbf{A}} \in \mathrm{RS}\,[L, \rho_{\mathbf{A}}]\,,\ \tilde{\pi}_{\mathbf{B}} \in \mathrm{RS}\,[L, \rho_{\mathbf{B}}]\,,\ \tilde{\pi}_{\mathbf{C}} \in \mathrm{RS}\,[L, \rho_{\mathbf{C}}]$$

Indeed, if any of the above conditions does not hold, then the weak soundness condition is immediately fulfilled by the violating oracle (see Lemma 4.4.7). Note that all the rates above equal $sT/|L|$.

Let $\tilde{f} \colon [T] \to \mathbb{F}^{rs}$ be the computation trace induced by the oracles sent by the prover, that is, $\tilde{f}(t) := \tilde{w}_1(t) \| \ldots \| \tilde{w}_r(t)$ for every $t \in [T]$, where each $\tilde{w}_i$ is the sub-trace encoded in $\tilde{\pi}_{w_i}$. By Definition 4.7.1, we know that, with probability at least $1 - \epsilon$ over the verifier's randomness $a = (a_1, \ldots, a_r) \in \mathbb{F}^{r\ell}$, either $\tilde{f}$ does not satisfy some time constraint or some boundary constraint. We analyze each of these two cases.

- *A time constraint is violated*, i.e., there exists t such that $\mathbf{A}(a)\tilde{f}(t,t+1) \circ \mathbf{B}(a)\tilde{f}(t,t+1) \neq \mathbf{C}(a)\tilde{f}(t,t+1)$.

  If the rowcheck condition is violated, then the interpolations $\hat{\pi}_{\mathbf{A}}, \hat{\pi}_{\mathbf{B}}, \hat{\pi}_{\mathbf{C}}$ of $\tilde{\pi}_{\mathbf{A}}, \tilde{\pi}_{\mathbf{B}}, \tilde{\pi}_{\mathbf{C}}$ are such that there exists $\alpha \in H_{\mathrm{ROW}}$ for which $\hat{\pi}_{\mathbf{A}}(\alpha) \cdot \hat{\pi}_{\mathbf{B}}(\alpha) - \hat{\pi}_{\mathbf{C}}(\alpha) \neq 0$, which means that $\hat{\pi}_{\mathbf{A}}\hat{\pi}_{\mathbf{B}} - \hat{\pi}_{\mathbf{C}}$ is not divisible by $\mathbb{Z}_{H_{\mathrm{ROW}}}$, and thus it is not the case that $\mathbf{\Pi}_{\mathrm{ROW}}[\tilde{\pi}_{\mathbf{A}}, \tilde{\pi}_{\mathbf{B}}, \tilde{\pi}_{\mathbf{C}}] \in \mathrm{RS}\,[L, \rho_{\mathrm{ROW}}]$.

  Otherwise, it must be the case that the lincheck condition is violated, which means that with probability at least $1 - \frac{3sT+1}{\mathbb{F}}$, one of the instances output by the lincheck verifier is not the prescribed code.

- *A boundary constraint is violated*, i.e., there exists $(t, j, \alpha) \in \mathcal{B}$ such that $\tilde{f}(t)_j \neq \alpha$.

  Let $i \in [r]$ and $j' \in \{1, \ldots, s\}$ be such that $j = j' + s \cdot (i-1)$, which means that $(t, j', \alpha) \in \mathcal{B}_i$ (see Item 5); note that $\tilde{f}(t)_j = \tilde{w}_i(t)_{j'}$. Suppose that $\mathbf{\Pi}_{\mathcal{B}_i}[\tilde{\pi}_{w_i}] \in \mathrm{RS}\,[L, \rho_{\mathcal{B}_i}]$ (otherwise we are done). Then, by definition of $\mathbf{\Pi}_{\mathcal{B}_i}[\tilde{\pi}_{w_i}]$, the interpolation of $\mathbf{\Pi}_{\mathcal{B}_i}[\tilde{\pi}_{w_i}]$ times the polynomial $\mathbb{Z}_{E_i}$ equals $\hat{\pi}_{w_i} - B_i$, where $\hat{\pi}_{w_i}$ is the interpolation of $\tilde{\pi}_{w_i}$. Using the fact that $\mathbb{Z}_{E_i}$ vanishes at $\mathcal{T}(t, j')$, we conclude that $\tilde{f}(t)_j = \tilde{w}_i(t)_{j'} = \hat{\pi}_{w_i}(\mathcal{T}(t, j')) = B_i(\mathcal{T}(t, j')) = \alpha$, which is a contradiction.

Efficiency. The oracle reduction in Protocol 4.7.7 adds a single round of interaction to the r-round interactive automaton at hand, and thus the round complexity is $r + 1$. The length of the reduction is determined by the $r + 1$ oracles that are sent in the interaction phase and the 3 oracles sent in the lincheck protocol; each oracle is of length $|L|$, and thus the total length is $(r + 4)|L|$. In each round there is one probe to the virtual oracles, and hence the locality is $r + 1$. The distance follows by invoking Lemma 4.4.7 with respect to the maximum rate of $4sT/|L|$ for the lincheck reduction, yielding distance $\frac{1}{2}(1 - 4sT/|L|)$. Finally, the prover and verifier time complexity follows immediately from the time complexity of the lincheck and rowcheck protocols.

4.8 Reducing machines to interactive automata

We define the *R1CS machines* relation, which is about checking algebraic computation with external memory. We then show how to reduce it, via an oracle reduction of linear length and locality 3, to testing proximity to the Reed-Solomon code (see Lemma 4.8.2 below). This reduction builds on the results from Section 4.7.

Loosely speaking, the R1CS machines relation asserts that a machines's *execution trace* and *memory trace* satisfy a rank-1 constraint system, i.e., each pair of consecutive rows in one of the traces satisfies an R1CS equation. Additionally, the relation ensures that the execution and memory traces are consistent by checking that they are *permutations of each other*.[9] The relation also includes boundary constraints to ensure, e.g., that the machine starts its computation in a valid initial state and halts in an accepting final state.

[9]The use of permutations to express machine computations dates back at least to the seminal work of Babai, Fortnow, Levin, and Szegedy [16], and originates in the study of nearly-linear time reductions among different computation models [100, 128].

In the following, we denote the computation time by T, the computation width by k, and the number of constraints in an R1CS matrix by m. Given a trace $f\colon [T] \to \mathbb{F}^k$, we denote by $f(i,j) \in \mathbb{F}^{2k}$ the concatenation of $f(i)$ and $f(j)$. We formally define the R1CS machines relation $\mathcal{R}_{\mathrm{R1M}}$ as follows.

Definition 4.8.1. *The relation $\mathcal{R}_{\mathrm{R1M}}$ of bounded accepting computations on* **R1CS machines** *consists of pairs $(\mathbb{x}, \mathbb{w})$ defined as follows. An instance $\mathbb{x} = (\mathbb{F}, (A,B,C), (A',B',C'), \mathcal{B})$ consists of a finite field $\mathbb{F}$, matrices $A,B,C \in \mathbb{F}^{m\times 2k}$ defining time constraints, matrices $A',B',C' \in \mathbb{F}^{m\times 2k}$ defining memory constraints, and a set of boundary constraints $\mathcal{B} \subseteq [T]\times[k]\times\mathbb{F}$. A witness $\mathbb{w} = (f,\pi)$ consists of an execution trace $f\colon [T]\to\mathbb{F}^k$ and permutation $\pi\colon [T]\to[T]$. A pair $(\mathbb{x},\mathbb{w})$ is in $\mathcal{R}_{\mathrm{R1M}}$ if the following holds.*

- Time constraints: $\forall\, t \in [T-1],\ \ Af(t,t+1) \circ Bf(t,t+1) = Cf(t,t+1)\,.$
- Memory constraints: $\forall\, t \in [T-1],\ \ A'f(\pi(t),\pi(t+1)) \circ B'f(\pi(t),\pi(t+1)) = C'f(\pi(t),\pi(t+1))\,.$
- Boundary constraints: $\forall\,(t,j,\alpha) \in \mathcal{B},\ \ f(t)_j = \alpha\,.$

The main result of this section is an oracle reduction, with linear length and locality 3, from R1CS machines to testing proximity to the Reed–Solomon code. The verifier uses $\mathrm{poly}(|\mathbb{x}|) = \mathrm{poly}(m,k,\log T)$ field operations, which is *exponentially* faster than the computation time T, since the dependence on the T is polylogarithmic instead of polynomial.

Lemma 4.8.2. *Let $\mathbb{F}$ be a finite field, H a coset in $\mathbb{F}$, and $L \subseteq \mathbb{F}$ with $L \cap H = \emptyset$. Let $\mathcal{T}\colon [N]\times[n]\to H$ be a trace embedding in $\mathbb{F}$. There is an RS oracle reduction over domain L (Protocol 4.8.6) for instances $\mathbb{x}$ of $\mathcal{R}_{\mathrm{R1M}}$ over $\mathbb{F}$, with computation time $T = N-1$, width $k \le n/2$, and $m \le n-2$ constraints. The reduction has 3 rounds of interaction and the following parameters:*

length	$6\lvert L\rvert$	*soundness error*	$(kT+3Nn+1)/\lvert\mathbb{F}\rvert$	*prover time*	$\lvert L\rvert\cdot(\log\lvert L\rvert + \lvert\mathbb{x}\rvert)$
locality	3	*distance*	$\frac{1}{2}(1-4Nn/\lvert L\rvert)$	*verifier time*	$\mathrm{poly}(\lvert\mathbb{x}\rvert)$

As discussed in Section 4.2.4, the main technical tool is a *matrix permutation check protocol*, which checks that two matrices are row permutations of one another. Our high-level strategy for proving Lemma 4.8.2 is to use the foregoing interactive protocol to check consistency between the execution and memory traces of an R1CS machine, and then check each of the traces via the oracle reduction for interactive R1CS automata.

Organization. In Section 4.8.1 we describe the matrix permutation check protocol, and explain how to represent it via an R1CS equation. In Section 4.8.2 we prove Lemma 4.8.2 by using the foregoing sub-protocol and the oracle reduction for interactive R1CS automata in Section 4.7.

4.8.1 Matrix permutation check protocol

We describe a protocol for checking that two matrices over a (sufficiently large) finite field $\mathbb{F}$ are row permutations of one another. The protocol leverages interaction with a prover to avoid more expensive tools that establish equivalence under permutations, such as sorting or routing networks. Since we ultimately wish to express the protocol via an interactive R1CS automaton (see Section 4.8.2), we present the protocol as a distribution over R1CS matrices. We note, however, that this protocol can also be formalized in other ways.

For notational convenience, we view a $T \times k$ matrix over $\mathbb{F}$ as a function $f\colon [T] \to \mathbb{F}^k$. Then $f'\colon [T] \to \mathbb{F}^k$ is a permutation of f if there is a permutation $\pi\colon [T] \to [T]$ such that $f'(t) = f(\pi(t))$ for all $t \in [T]$.

From permutation to identity testing. Checking that two matrices are row permutations of one another can be expressed as a polynomial identity testing problem, as the permutation condition is an equality problem between multi-sets over vectors. We can encode each vector $v \in \mathbb{F}^k$ as a univariate polynomial $q_v(Y) := \sum_{j=1}^{k} Y^j v_j$, and encode a multi-set of vectors $S = \{v^{(t)}\}_{t\in[T]}$ as a bivariate polynomial $q_S(X,Y) := \prod_{t=1}^{T}(X - q_{v^{(t)}}(Y))$. Then, two multi-sets S and S' are equal if and only if $q_S(X,Y) \equiv q_{S'}(X,Y)$.

This suggests a probabilistic protocol to check the permutation condition. The verifier sends to the prover two random elements $\alpha, \beta \in \mathbb{F}$. Then, the prover has to convince the verifier that $q_S(\alpha,\beta) = q_{S'}(\alpha,\beta)$. This suffices since if $q_S(X,Y) \equiv q_{S'}(X,Y)$, then $q_S(\alpha,\beta) = q_{S'}(\alpha,\beta)$ with probability 1 over α, β; if instead $q_S(X,Y) \not\equiv q_{S'}(X,Y)$, then $q_S(\alpha,\beta) \neq q_{S'}(\alpha,\beta)$ with probability at least $1 - kT/|\mathbb{F}|$ over α, β.

Intuitively, evaluation at β plays the role of a hash function: if two vectors $v, u \in \mathbb{F}^k$ are not equal then, with probability at least $1 - k/|\mathbb{F}|$ over β, it holds that $q_v(\beta) \neq q_u(\beta)$. This "collapses" all vectors in S and S' to single field element such that, with high probability, distinct vectors hash to distinct elements. Evaluation at α plays the role of another hash function, except that this time it is with respect to the vectors of all hashes, namely $(q_v(\beta))_{v\in S}$ and $(q_u(\beta))_{u\in S'}$. After these two hash function evaluations, only two elements need to be compared, namely $q_S(\alpha,\beta)$ and $q_{S'}(\alpha,\beta)$.

Probabilistic checks for multi-set equality like the above ones are familiar techniques from program checking [114, 47], and have been recently applied to check machine computations (see, e.g., [155]).

From identity testing to a protocol. We need to design a protocol that enables a prover to convince the verifier that a function $f'\colon [T] \to \mathbb{F}^k$ is a permutation of another function $f\colon [T] \to \mathbb{F}^k$. The discussion so far tells us that it suffices for the verifier to learn the random evaluation of bivariate polynomials related to f and f', but does not tell us what protocol to run.

Towards this end, consider the bivariate polynomials $\{\chi_t(X,Y)\}_{t\in[T]}$ defined as follows:

$$\chi_t(X,Y) := \prod_{i=1}^{t}\left(X - \sum_{j=1}^{k} Y^j f(i)_j\right) - \prod_{i=1}^{t}\left(X - \sum_{j=1}^{k} Y^j f'(i)_j\right) \quad . \tag{4.2}$$

Observe that $\chi_T \equiv 0$ if and only if there exists a permutation π such that $f'(t) = f(\pi(t))$ for all $t \in [T]$.

For any choice of $\alpha, \beta \in \mathbb{F}$, we consider an auxiliary trace $g\colon [T+1] \to \mathbb{F}^3$ that "incrementally computes" $\chi_T(\alpha, \beta)$ as follows. The first and second columns of g are defined so that $g(1)_1 = g(1)_2 = 1_{\mathbb{F}}$ and, for $1 < t \leq [T+1]$, $g(t)_1$ and $g(t)_2$ respectively contain the first and second terms of $\chi_{t-1}(\alpha, \beta)$:

$$g(t)_1 := \prod_{i=1}^{t-1} \left(\alpha - \sum_{j=1}^{k} \beta^j f(i)_j \right) = g(t-1)_1 \cdot \left(\alpha - \sum_{j=1}^{k} \beta^j f(t-1)_j \right) \ ,$$
$$g(t)_2 := \prod_{i=1}^{t-1} \left(\alpha - \sum_{j=1}^{k} \beta^j f'(i)_j \right) = g(t-1)_2 \cdot \left(\alpha - \sum_{j=1}^{k} \beta^j f'(t-1)_j \right) \ .$$

Note that $g(t)_1$ and $g(t)_2$ can be derived from $g(t-1)_1$ and $g(t-1)_2$ respectively. The third column of g is the difference of the first two columns: for every $t \in [T+1]$ we define $g(t)_3 = g(t)_1 - g(t)_2 = \chi_{t-1}(\alpha, \beta)$. Observe that, if $\chi_T \not\equiv 0$ then $g(T+1)_3 = \chi_T(\alpha, \beta) = 0$ with probability at most $kT/|\mathbb{F}|$.

We can summarize the above via a statement that involves local constraints among adjacent variables.

Lemma 4.8.3. *Given $f, f'\colon [T] \to \mathbb{F}^k$, define the probability*

$$\mu(f, f') := \Pr_{\alpha,\beta \leftarrow \mathbb{F}} \left[\begin{array}{l} \exists\, g\colon [T+1] \to \mathbb{F}^3 \text{ such that} \\ \bullet\ g(1)_1 = g(1)_2 = 1 \text{ and } g(T+1)_3 = 0 \\ \bullet\ \forall\, t \in [T]\,,\ g(t+1)_1 = g(t)_1 \cdot (\alpha - \sum_{j=1}^{k} \beta^j f(t)_j) \\ \bullet\ \forall\, t \in [T]\,,\ g(t+1)_2 = g(t)_2 \cdot (\alpha - \sum_{j=1}^{k} \beta^j f'(t)_j) \\ \bullet\ \forall\, t \in [T+1]\,,\ g(t)_3 = g(t)_1 - g(t)_2 \end{array} \right] .$$

Then the following conditions hold.

- Completeness: *if f' is a permutation of f then $\mu(f, f') = 1$.*
- Soundness: *if f' is not a permutation of f then $\mu(f, f') \leq kT/|\mathbb{F}|$.*

The protocol via an R1CS equation. We re-write Lemma 4.8.3 in the language of R1CS equations, so that in Section 4.8.2 we can embed the matrix permutation check protocol in an interactive R1CS automaton.

For functions $a\colon [T] \to \mathbb{F}^k$ and $b\colon [T'] \to \mathbb{F}^{k'}$, we denote by $a \| b\colon [\max(T, T')] \to \mathbb{F}^{k+k'}$ the function defined as $(a\|b)(t) := a(t)\|b(t)$, where we pad a or b via all-zero rows if $T \neq T'$. We construct a distribution of R1CS matrices such that: if f' is a permutation of f, then there exists an auxiliary trace g such that $f\|f'\|g$ satisfies the constraints of the R1CS matrices, and otherwise, with high probability, there is no auxiliary trace g that makes $f\|f'\|g$ satisfy the constraints of the R1CS matrices.

Lemma 4.8.4. *Let $T, k, n \in \mathbb{N}$ with $n \geq 2k$. Let $\mathsf{0}\colon [T] \to \mathbb{F}^{n-2k}$ always output zeros. There exists a polynomial-time samplable distribution $\mathcal{D}$ over tuples of matrices $(A_{\mathrm{p}}, B_{\mathrm{p}}, C_{\mathrm{p}}) \in$*

$(\mathbb{F}^{3\times 4n})^3$*, where drawing a sample from* $\mathcal{D}$ *requires two random elements in* $\mathbb{F}$*, such that for every* $f, f' \colon [T] \to \mathbb{F}^k$ *the probability*

$$\mu(f,f') := \Pr_{(A_{\mathrm{p}},B_{\mathrm{p}},C_{\mathrm{p}})\leftarrow\mathcal{D}}\left[\begin{array}{c}\exists\, g \colon [T+1]\to\mathbb{F}^n \text{ with } g(1)_1 = g(1)_2 = 1 \text{ and } g(T+1)_3 = 0 \text{ s.t.}\\ \text{letting } h := f\|f'\|0\|g\,,\\ \forall\, t\in[T]\,,\ A_{\mathrm{p}}h(t,t+1)\circ B_{\mathrm{p}}h(t,t+1) = C_{\mathrm{p}}h(t,t+1)\end{array}\right],$$

satisfies the following conditions.

- Completeness: *if* f' *is a permutation of* f*, then* $\mu(f,f') = 1$.
- Soundness: *if* f' *is not a permutation of* f*, then* $\mu(f,f') \le kT/|\mathbb{F}|$.

Proof. We construct $(A_{\mathrm{p}}, B_{\mathrm{p}}, C_{\mathrm{p}})$ so that, for $h := f\|f'\|0\|g$ with $g(1)_1 = g(1)_2 = 1$ and $g(T+1)_3 = 0$,

$$\{A_{\mathrm{p}}h(t,t{+}1)\circ B_{\mathrm{p}}h(t,t{+}1) = C_{\mathrm{p}}h(t,t{+}1)\}_{t\in[T]} \longrightarrow \{g(t)_3 = g(t)_1 - g(t)_2 = \chi_{t-1}(\alpha,\beta)\}_{t\in[T]}\ .$$

Viewing $h(t,t+1)$ as a vector

$$u^{(1)}\|v^{(1)}\|0^{n-2k}\|w^{(1)}\|u^{(2)}\|v^{(2)}\|0^{n-2k}\|w^{(2)} \in \mathbb{F}^{4n}\ ,$$

for $u^{(1)}, u^{(2)}, v^{(1)}, v^{(2)} \in \mathbb{F}^k$ and $w^{(1)}, w^{(2)} \in \mathbb{F}^{2k}$, we choose the matrices to enforce the following rank-1 constraints:

$$\begin{aligned} w_1^{(1)} \cdot \left(\alpha - \sum_{j=1}^{k} \beta^j u_j^{(2)}\right) &= w_1^{(2)}\ ,\\ w_2^{(1)} \cdot \left(\alpha - \sum_{j=1}^{k} \beta^j v_j^{(2)}\right) &= w_2^{(2)}\ ,\\ w_1^{(2)} - w_2^{(2)} &= w_3^{(2)}\ . \end{aligned}$$

Note that $A_{\mathrm{p}}, B_{\mathrm{p}}, C_{\mathrm{p}}$ are $3 \times 4n$ matrices over $\mathbb{F}$, as claimed. □

4.8.2 Proof of Lemma 4.8.2

We provide an oracle reduction from checking the R1CS machines relation to testing proximity to the Reed-Solomon code. We rely on two ingredients: (a) the matrix permutation check protocol (Lemma 4.8.4), and (b) the oracle reduction from the R1CS automata relation to testing the Reed-Solomon code (Lemma 4.7.2).

Observe that checking membership in the relation $\mathcal{R}_{\mathrm{R1M}}$ amounts to checking that: (1) the execution trace f satisfies the time constraints; (2) the memory trace f' satisfies the memory constraints; (3) the memory trace f' is a permutation of the execution trace f; and (4) the execution trace f satisfies the boundary constraints. We "program" an automaton to check time constraints and memory constraints, and additionally program the (interactive) automaton to

check the permutation condition via the matrix permutation check protocol in Section 4.8.1. This latter is the only place wherein we use the interactivity of the automata.

We now elaborate on this plan, by first describing the reduction from machines to interactive automata, and then describing the oracle reduction induced by it.

From machines to interactive automata. Let $\mathbb{x} = (\mathbb{F}, (A, B, C), (A', B', C'), \mathcal{B})$ be an instance for the relation $\mathcal{R}_{\mathrm{R1M}}$ (see Definition 4.8.1), in which the time constraints are matrices $A, B, C \in \mathbb{F}^{m \times 2k}$, the memory constraints are matrices $A', B', C' \in \mathbb{F}^{m \times 2k}$, and the boundary constraints are a set $\mathcal{B} \subseteq [T] \times [k] \times \mathbb{F}$.

Below we describe how to construct an instance $\mathbb{x}' = (\mathbb{F}, \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathcal{B}', \ell = 2)$ for the relation $\mathcal{R}^{2,\epsilon}_{\mathrm{R1A}}$ with $\epsilon := Nn/|\mathbb{F}|$ (see Definition 4.7.1). After that we describe how to transform a witness for $\mathbb{x}$ into one for $\mathbb{x}'$.

- **Instance reduction.** First we split each matrix $M \in \{A, B, C, A', B', C'\}$ into halves $M_1, M_2 \in \mathbb{F}^{m \times k}$ by putting the first k columns into M_1 and other k into M_2. Now let $A_{\mathrm{p}}(\alpha, \beta), B_{\mathrm{p}}(\alpha, \beta), C_{\mathrm{p}}(\alpha, \beta) \in \mathbb{F}^{3 \times 4n}$ be the matrices obtained from the distribution $\mathcal{D}$ in Lemma 4.8.4 with randomness $\alpha, \beta \in \mathbb{F}$.

  We now define the function $\mathbf{A} \colon \mathbb{F}^2 \to \mathbb{F}^{n \times 4n}$.

$$\mathbf{A}(\alpha, \beta) := \begin{array}{|c|c|c|c|c|c|c|c|} \hline A_1 & 0^{m \times k} & 0^{m \times n-2k} & 0^{m \times n} & A_2 & 0^{m \times k} & 0^{m \times n-2k} & 0^{m \times n} \\ \hline 0^{m \times k} & A'_1 & 0^{m \times n-2k} & 0^{m \times n} & 0^{m \times k} & A'_2 & 0^{m \times n-2k} & 0^{m \times n} \\ \hline \multicolumn{8}{|c|}{A_{\mathrm{p}}(\alpha, \beta)} \\ \hline \multicolumn{8}{|c|}{0^{(n-2m-3) \times 4n}} \\ \hline \end{array}$$

  We similarly define the functions $\mathbf{B}, \mathbf{C} \colon \mathbb{F}^2 \to \mathbb{F}^{n \times 4n}$.[10]

  Observe that we have constructed the functions above so that, given a trace $h \colon [T+1] \to \mathbb{F}^{2n}$ parsed as the concatenation of traces $f \colon [T] \to \mathbb{F}^k$, $f' \colon [T] \to \mathbb{F}^k$, $g \colon [T+1] \to \mathbb{F}^n$ (discarding columns as appropriate), if

$$\forall\, t \in [T]\,,\ \mathbf{A}(\alpha, \beta) h(t, t+1) \circ \mathbf{B}(\alpha, \beta) h(t, t+1) = \mathbf{C}(\alpha, \beta) h(t, t+1)\ ,$$

  then we know that f satisfies time constraints, f' satisfies memory constraints, and h satisfies the constraints in Lemma 4.8.4 induced by the randomness α, β.

  We define the boundary constraints $\mathcal{B}' \subseteq [T+1] \times [4n] \times \mathbb{F}$ to be the union of the boundary constraints $\mathcal{B}$ in $\mathbb{x}$ and the boundary constraints from the matrix permutation check (in Lemma 4.8.4). More precisely, $(t, j, \alpha) \in \mathcal{B}'$ if and only if $(t, j, \alpha) \in \mathcal{B}$ or $(t, j, \alpha) \in \{(1, n+1, 1), (1, n+2, 1), (T+1, n+3, 0)\}$.

  Note that transforming $\mathbb{x}$ into $\mathbb{x}'$ can be performed in linear time.

[10] The functions $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are actually supposed to be from $\mathbb{F}^{r\ell}$ to $\mathbb{F}^{n \times 2rn}$ where r is the number of rounds and ℓ is the number of field elements sent by the verifier in each round (see Definition 4.7.1). But here the verifier's first message is empty and its second message has two field elements. So, given that $r = 2$, we find it more convenient to take $\mathbf{A}, \mathbf{B}, \mathbf{C}$ to be functions from $\mathbb{F}^2$ to $\mathbb{F}^{n \times 4n}$.

- **Witness reduction.** Suppose that $\mathbb{x}$ has a valid witness $\mathbb{w} = (f, \pi)$, namely, $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathrm{R1M}}$. Let $f' \colon [T] \to \mathbb{F}^k$ be the memory trace obtained by permuting f according to π: for every $t \in [T]$, we define $f'(t) := f(\pi(t))$. Let $P_{\mathbb{w}'}$ be the prover strategy for the game defined by $\mathbb{x}'$ that works as follows:
  1. the prover sends $(f \| f' \| 0) \colon [T] \to \mathbb{F}^{2n}$, the padded concatenation of the execution and memory traces;
  2. the prover receives two elements $\alpha, \beta \in \mathbb{F}$ from the verifier;
  3. the prover uses α, β to construct an auxiliary trace $g \colon [T+1] \to \mathbb{F}^n$ such that $\mu(f, f') = 1$ (as guaranteed by Lemma 4.8.4);
  4. the prover sends g.

Observe that the foregoing is a (standard, polynomial-time) reduction from $\mathcal{R}_{\mathrm{R1M}}$ to $\mathcal{R}_{\mathrm{R1A}}^{2,\epsilon} =: (\mathcal{R}_{\mathsf{Yes}}, \mathcal{L}_{\mathsf{No}})$.

Claim 4.8.5. *If* $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathrm{R1M}}$*, then* $(\mathbb{x}', P_{\mathbb{w}'}) \in \mathcal{R}_{\mathsf{Yes}}$*. If instead* $\mathbb{x} \notin \mathcal{L}(\mathcal{R}_{\mathrm{R1M}})$*, then* $\mathbb{x}' \in \mathcal{L}_{\mathsf{No}}$*.*

Proof. Suppose that $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathrm{R1M}}$. Then $(\mathbb{x}', P_{\mathbb{w}'}) \in \mathcal{R}_{\mathsf{Yes}}$ because by construction the R1CS automaton defined by $\mathbb{x}'$ runs the permutation check on f, f' (which always passes), time constraints check on f (which always passes), and memory constraints check on f' (which always passes).

Suppose instead that $\mathbb{x} \notin \mathcal{L}(\mathcal{R}_{\mathrm{R1M}})$. We argue that $\mathbb{x}' \in \mathcal{L}_{\mathsf{No}}$. Consider a candidate prover strategy $P_{\mathbb{w}'}$. We need to argue that $P_{\mathbb{w}'}$ wins the game defined by $\mathbb{x}'$ with probability at most $\epsilon = kT/|\mathbb{F}|$. Let $(f \| f') \colon [T] \to \mathbb{F}^{2k}$ be the first message sent by $P_{\mathbb{w}'}$. If f does not satisfy the time constraints in $\mathbb{x}$ or f' does not satisfy the memory constraints in $\mathbb{x}$, then the prover loses with probability 1, because the R1CS automaton $\mathbb{x}'$ checks both of these conditions. So suppose that f and f' satisfy the time and memory constraints respectively. This means that f' is *not* a permutation of f (for otherwise $\mathbb{x}$ would have been in the language of $\mathcal{R}_{\mathrm{R1M}}$), and so we can use the soundness condition of Lemma 4.8.4. Indeed, we know that, with probability at least $1 - kT/|\mathbb{F}|$ over the verifier's choice of $\alpha, \beta \in \mathbb{F}$, the second message $g \colon [T+1] \to \mathbb{F}^3$ of the prover does *not* satisfy the permutation check constraints, in which case the prover loses. □

Protocol 4.8.6. Let $\mathbb{F}$ be a finite field, H a coset in $\mathbb{F}$, and $L \subseteq \mathbb{F}$ with $L \cap H = \emptyset$. Let $\mathcal{T} \colon [N] \times [n] \to H$ be a trace embedding in $\mathbb{F}$ with $N = T + 1$ and $n \geq 2k$. We need to construct an oracle reduction (P, V) that works for instances $\mathbb{x} = (\mathbb{F}, (A, B, C), (A', B', C'), \mathcal{B})$ of computation time T and width k.

The oracle reduction is straightforward: we reduce the R1CS machine to an interactive R1CS automaton and then invoke the oracle reduction (P', V') from Lemma 4.7.2. (Note that the hypothesis in Lemma 4.8.2 is the same as in Lemma 4.7.2, so that we can indeed invoke the latter.)

In more detail, the prover P and verifier V each transform the given instance $\mathbb{x}$ for the relation $\mathcal{R}_{\mathrm{R1M}}$ into the instance $\mathbb{x}'$ for the relation $\mathcal{R}_{\mathrm{R1A}}^{2,\epsilon} = (\mathcal{R}_{\mathsf{Yes}}, \mathcal{L}_{\mathsf{No}})$, following the instance reduction described above. Also, the prover P transforms a witness $\mathbb{w} = (f, \pi)$ for $\mathbb{x}$ into a witness $P_{\mathbb{w}'}$ for $\mathbb{x}'$, following the witness reduction described above. Then, letting the prover P and V engage in an oracle reduction with P playing the role of $P'(\mathbb{x}', P_{\mathbb{w}'})$ and V playing the role of $V'(\mathbb{x}')$. Finally, the verifier V outputs whatever V' outputs.

Completeness. If $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathrm{R1M}}$, then $(\mathbb{x}', P_{\mathbb{w}'}) \in \mathcal{R}_{\mathsf{Yes}}$, and completeness of the oracle reduction (P', V') for automata implies completeness of the oracle reduction (P, V) for machines.

Soundness. If $\mathbb{x} \notin \mathcal{L}(\mathcal{R}_{\mathrm{R1M}})$, then $\mathbb{x} \in \mathcal{L}_{\mathsf{No}}$, and so we obtain the soundness guarantee of the oracle reduction (P', V').

Efficiency. The prover P runs in time $\mathrm{poly}(|\mathbb{x}| + T)$ and the verifier V runs in time $\mathrm{poly}(|\mathbb{x}|)$ because the respective running times in the oracle reduction (P', V') are $\mathrm{poly}(|\mathbb{x}'| + N)$ and the verifier V runs in time $\mathrm{poly}(|\mathbb{x}'|)$ and it holds that $|\mathbb{x}'| = O(|\mathbb{x}|)$ and $N = O(T)$. (Also, $\mathbb{x}'$ can be efficiently derived from $\mathbb{x}$.)

The locality of (P, V) is 3 because the locality of (P', V') is $r + 1$ when the interactive automaton has r rounds, which in the case of $\mathbb{x}'$ is $r = 2$. The length of (P, V) is $6|L|$.

4.9 Proofs of main results

In Section 4.9.1 we prove Theorem 3, and in Section 4.9.2 we prove Theorem 2.

4.9.1 Checking satisfiability of algebraic machines

In Section 4.8 we obtained an oracle reduction from the R1CS machine relation to testing proximity to the Reed–Solomon code. We now combine this oracle reduction with a linear-size IOP of proximity for the Reed–Solomon code of [25] to obtain our main result. We first recall this latter result.

Lemma 4.9.1 ([25, Theorem 5.1])**.** *Fix a rate parameter $\rho \in (0, 1)$ and a proximity parameter $\delta \in (0, (1 - \rho)/2)$. Let $\mathbb{F}$ be a finite field, L_0 a subgroup of $\mathbb{F}$ that itself has a subgroup of size $\Theta(|L_0|^\alpha)$ for some $\alpha \in (0, 1)$, and let L be a coset of L_0. There exists a 2-round IOPP system for $\mathrm{RS}[L, \rho]$ with linear proof length, $\mathsf{q}_{\mathsf{w}} = 1$, $\mathsf{q}_\pi = 2$, distance parameter δ, constant soundness error, and constant query complexity. The prover uses $O(|L| \operatorname{polylog} |L|)$ field operations and the verifier uses $\mathrm{polylog}(|L|)$ field operations. The verifier's first message is empty.*

The result in [25] is stated with soundness $1/2$ and some constant query complexity; applying query reduction to the protocol's second round yields the same result but with 3 queries and constant soundness.

Next we introduce our definition for *large smooth fields*, which captures the properties that we use to construct suitable trace embeddings. When a field is $(T(n), k(n), \rho(n))$-smooth according to the definition below, we can use the algorithm of Lemma 4.5.2 to construct a trace embedding of size $T(n) \times O(k(n))$ in time $\mathrm{poly}(\log T(n), k(n))$. For example, the family $\{\mathbb{F}_{p^{2^n}}\}_{n \in \mathbb{N}}$ is $(p^n, O(n), O(1))$-smooth; the same is true of fields with smooth multiplicative subgroups in the sense of [41]. The additional, and somewhat technical, conditions in the definition ensure the existence of a subgroup L_0 of $\mathbb{F}$ of size $\Theta(k(n) \cdot T(n))$ with a suitable coset L, so that we can invoke Lemma 4.9.1 with rate parameter $\rho(n)$.

Definition 4.9.2. *A family of fields $\{\mathbb{F}(n)\}_{n \in \mathbb{N}}$ is $(T(n), k(n), \rho(n))$-**smooth** if there exists $\alpha \in (0, 1)$ and a family $\{H_1(n), H_2(n), L_0(n)\}_{n \in \mathbb{N}}$, such that $H_1(n), H_2(n), L_0(n)$ are subgroups of $\mathbb{F}(n)$, where for all n:*

- $|H_1(n)| = T(n)$, $k(n) \leq |H_2(n)| = O(k(n))$, *and* $|H_1(n) \cap H_2(n)| = 1$*;*
- $L_0(n)$ *has a subgroup of size* $\Theta(|L_0(n)|^\alpha)$*;*
- *if* $H(n)$ *is the smallest subgroup of* $\mathbb{F}(n)$ *containing* $H_1(n)$ *and* $H_2(n)$ *then* $L_0(n)$ *contains* $H(n)$ *with* $\rho(n) \geq |H(n)|/|L_0(n)| = \Omega(\rho(n))$.

Note that if $\{\mathbb{F}(n)\}_{n\in\mathbb{N}}$ is $(T(n), k(n), \rho(n))$-smooth then it is also $(T(n), ck(n), c'\rho(n))$-smooth for any constants $c \in (0, 1]$, $c' > 1$.

The above condition suffices for the construction to be feasible, but for prover efficiency we require much more structure. The following condition guarantees the existence of a fast Fourier transform which runs in time $O(n \log n)$; it is a usual notion of smoothness for integers. Again, ensembles of binary fields or fields with smooth multiplicative subgroups satisfy this definition. Crucially for us, if our field family satisfies the following condition then the prover time in Lemma 4.9.1 can be reduced to $O(|L| \log |L|)$.

Definition 4.9.3. *A family of fields* $\{\mathbb{F}(n)\}_{n\in\mathbb{N}}$ *is* $(T(n), k(n), \rho(n))$**-very smooth** *if it is* $(T(n), k(n), \rho(n))$*-smooth and there exists a constant* c *such that the prime factors of* $|L_0(n)|$ *are all at most* c *for all* n*.*

To give the formal statement of our main theorem, we first define a parameterized version of the R1CS machine relation.

Definition 4.9.4. *The relation* $\mathcal{R}_{\mathrm{R1M}}[\mathbb{F}(n), T(n), k(n)]$ *consists of pairs* $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}_{\mathrm{R1M}}$ *such that* $\mathbb{F} = \mathbb{F}(n)$*,* $T = T(n)$ *and* $k \leq k(n)$*.*

We derive our main result (informally stated in Theorem 3) by combining Lemma 4.8.2 and Lemma 4.9.1. We assume that $\mathbb{F}(n)$ is uniformly specified via a primitive element, and also via the factorization of $|\mathbb{F}(n)|$ and $|\mathbb{F}(n)^*|$ so that we can efficiently construct any (additive or multiplicative) subgroup of $\mathbb{F}(n)$.

Theorem 4.9.5. *Let* $\mathcal{F} = \{\mathbb{F}(n)\}_{n\in\mathbb{N}}$ *be a* $(T(n) + 1, 2k(n), \rho(n))$*-smooth field family with* $T(n) \geq n$*,* $k(n) = \mathrm{poly}(n)$*; let* $S(n) := k(n)T(n)/\rho(n)$*. There exists a universal constant* ϵ_0 *such that there exists a* 5*-round IOP for* $\mathcal{R}_{\mathrm{R1M}}[\mathbb{F}(n), T(n), k(n)]$ *with a proof length of* $O(S(n))$ *field elements,* 5 *queries, and soundness error* ϵ_0*. The verifier uses* $\mathrm{poly}(n, \log T(n))$ *field operations.*

Moreover, if $\mathcal{F}$ *is very smooth then the prover uses* $O(S(n)(\log S(n) + n))$ *field operations.*

Proof. Let $\mathbb{x}$ be an instance of the R1CS machine relation $\mathcal{R}_{\mathrm{R1M}}[\mathbb{F}(n), T(n), k(n)]$ (see Definition 4.8.1). Since $\mathbb{F}(n)$ comes from a smooth family, we can use Lemma 4.5.2 to efficiently construct a trace embedding $\mathcal{T}\colon H(n) \to H_1(n) \times H_2(n)$ with $|H_1(n)| = T(n) + 1$ and $2k(n) \leq |H_2(n)| = O(k(n))$; these choices, from the theorem's hypothesis, are compatible with Lemma 4.8.2.

Moreover, let $L_0(n)$ be as guaranteed by the smoothness condition, and let $L(n)$ be a coset of $L_0(n)$ which is not $L_0(n)$ itself. Since $H(n) \subseteq L_0(n)$, we have $L(n) \cap H(n) = \varnothing$ (as required by Lemma 4.8.2).

By Lemma 4.8.2, there exists a 3-round RS oracle reduction over domain L from $\mathcal{R}_{\mathrm{R1M}}$. By Lemma 4.9.1, and the smoothness condition, there exists a 2-round IOP of proximity for $\mathcal{R}_{\mathrm{RS}}$ over L. Applying Corollary 4.4.9 to these components yields a 5-round IOP for $\mathcal{R}_{\mathrm{RS}}$ with the stated parameters. □

4.9.2 Checking satisfiability of succinct arithmetic circuits

In this section we prove our result for the relation Succinct-ASAT, which consists of succinctly-represented arithmetic circuits that are satisfiable. We begin by defining this relation.

Definition 4.9.6. *Let $m \in \mathbb{N}$, $E\colon \mathbb{F}^m \to \mathbb{F}^m$ be an arithmetic circuit, $o \in \mathbb{F}^m$, and $T \in \mathbb{N}$. We define $H_{E,o,T}$ to be the set $\{a_1, \ldots, a_T\} \subseteq \mathbb{F}^m$, where $a_1 := o$ and, for all $i \in \{1, \ldots, T-1\}$, $a_{i+1} := E(a_i)$. In other words, E* **enumerates** *the set $H_{E,o,T}$ as o, $E(o)$, $E(E(o))$, and so on.*

Definition 4.9.7. *The relation* Succinct-ASAT *consists of pairs $\big((\mathbb{F}, m, E, o, T, I, D), w\big)$, where $\mathbb{F}$ is a finite field, $m \in \mathbb{N}$, $E\colon \mathbb{F}^m \to \mathbb{F}^m$ is an arithmetic circuit for enumerating gates, $o \in \mathbb{F}^m$ is the label of the output gate, $T \in \mathbb{N}$ is the number of gates, I is a subset of $H_{E,o,T}$ representing the input gates, $D\colon \mathbb{F}^m \to (\{+, \times\} \times H_{E,o,T} \times H_{E,o,T}) \cup \{\mathbb{F}\}$ is an arithmetic circuit that describes an arithmetic circuit C, and the witness $w\colon I \to \mathbb{F}$ is such that $C(w) = 0$.*

We define a parameterized version of Succinct-ASAT.

Definition 4.9.8. *The relation* Succinct-ASAT$[\mathbb{F}(n), T(n), k(n)]$ *consists of pairs $(\mathbb{x}, \mathbb{w}) \in$* Succinct-ASAT *such that $\mathbb{F} = \mathbb{F}(n)$, $T = T(n)$ and $|E| + |D| \leq k(n)$.*

Next we give the formal statement of Theorem 2.

Theorem 4.9.9. *There exist universal constants $\epsilon_0 \in (0,1)$, $c \in \mathbb{N}$ such that for any $(T(n)+1, ck(n), O(1))$-very smooth field family with $T(n) \geq n$, there is a 5-round IOP for* Succinct-ASAT$[\mathbb{F}(n), T(n), k(n)]$ *with a proof length of $O(S)$ field elements for $S := T \cdot k(n)$, 5 queries, and soundness error ϵ_0. The prover uses $O(S \log S)$ field operations and the verifier uses $\mathrm{poly}(|E|, |D|, k(n))$.*

Above, $|E|, |D|$ are the number of gates in E, D respectively. Note that C has T gates whose names are in $\mathbb{F}^m$, and hence the number of field elements needed to represent C is $\Theta(T \cdot m)$, because the representation includes for each gate in $H_{E,o,T} \subseteq \mathbb{F}^m$ the names of the gates to which that gate is connected. In particular, the proof length in Theorem 4.9.9 is linear in the number of field elements to represent C when $|E|, |D| = O(m)$.

The proof of Theorem 4.9.9 is a direct implication of the following lemma.

Lemma 4.9.10. *There exists a universal constant $c \in N$ such that there is a polynomial-time reduction from* Succinct-ASAT$[\mathbb{F}(n), T(n), k(n)]$ *to $\mathcal{R}_{\mathrm{R1M}}[\mathbb{F}(n), T(n), ck(n)]$. The size of the $\mathcal{R}_{\mathrm{R1M}}$ instance is linear in the size of the* Succinct-ASAT *instance.*

Proof sketch. We begin by describing the witness reduction. Let $\big((\mathbb{F}, m, E, o, T, I, D), w\big) \in$ Succinct-ASAT. The witness w assigns a value to every wire of C.

- The first part of the witness for the algebraic machine is a function $f\colon [3T] \times [k] \to \mathbb{F}$, where $k = c(|E| + |D|)$, as follows. For each gate $g \in H_{E,o,T}$, ordered by E starting from o, we add three rows corresponding to g, labelled l, r, o respectively. These rows include the gate label g and the labels and values of the input wires.
- The second part of the witness f' lists, for each gate g ordered by E starting from o, all the rows in which it appears in f: first the unique row labelled o corresponding to g, then all rows labelled l where g is a left input, then all rows labelled r where g is a right input.

The label l, r, o determines uniquely, for each row, to which gate it belongs in f'. Hence f' is a permutation of f; this permutation π is the last part of the witness. The whole witness consists of $O(T \cdot (|E| + |D|))$ field elements.

Now we discuss the instance reduction. The time constraints check that f is ordered correctly according to E, each gate g is correctly evaluated according to D, and that the left, right and output entries for g are all present. The memory constraints check that f' is ordered correctly according to E, and that the assignment of the g-wire in each row is consistent with the value of the gate g. Clearly the size of this constraint system is linear in $|E| + |D|$.

Finally, we have a boundary constraint that checks that the output of C is 0. □

Chapter 5

FRACTAL: transparent, post-quantum recursive composition

5.1 Introduction

Recursive composition. The time to validate a SNARG can be exponentially faster than the time to run the non-deterministic computation that it attests to, a property known as succinct verification. This exponential speedup raises an interesting prospect: could one produce a SNARG about a computation that involves validating prior SNARGs? Thanks to succinct verification, the time to run this (non-deterministic) computation would be essentially independent of the time of the prior computations. This *recursive composition* of SNARGs enables *incrementally verifiable computation* [144] and *proof-carrying data* [68, 45]. A critical technicality here is that, for recursive composition to work, the SNARG must be an *argument of knowledge*, i.e., a SNARK. This is because the security of a SNARG holds only against efficient adversaries, and the knowledge property ensures that prior SNARGs must have been efficiently produced, and so we can rely in turn on their security. A formal treatment of this can be found in [45], which discusses how the "strength" of a SNARG's knowledge property relates to how many recursions the SNARG supports.

Efficient recursion. Theory tells us that *any* succinct-verifier SNARK is recursively composable [45]. In practice, however, recursive composition is exceedingly difficult to realize efficiently. The reason is that, even if we have a SNARK that is concretely efficient when used "standalone", it is often prohibitively expensive to express the SNARK verifier's computation through the language supported by the SNARK. Indeed, while by now there are numerous SNARK constructions with remarkable concrete efficiency, *to date there is only a single efficient approach to recursion*. The approach, due to [33], uses pairing-based SNARKs with a special algebraic property discussed below.[1] This has enabled real-world applications such as Coda [121], a cryptocurrency that uses recursive composition to achieve strong scalability properties.

Limitations. The above efficient approach to recursion suffers from significant limitations.

[1] Bowe, Grigg, and Hopwood [54] propose an alternative approach for recursion that does not require the SNARK to have succinct verification. We refer the interested reader to [57], which develops theoretical foundations for this approach in detail.

- *It is pre-quantum.* Pairing-based SNARKs rely (at least) on the hardness of extracting discrete logarithms, and so are insecure against quantum attacks. Hence the approach of [33] is also insecure against quantum attacks. Devising an efficient *post-quantum* approach to recursion is an open problem.
- *It introduces toxic waste.* All known pairing-based SNARKs that can be used in the approach of [33] rely on a structured reference string (SRS). Sampling the SRS involves secret values (the "toxic waste") that must remain secret for security. Ensuring that this is the case in practice is difficult: the SRS must be sampled by some trusted party or via a cryptographic ceremony [31, 52, 53, 3]. Devising an efficient *transparent* (toxic-waste free) approach to recursion is an open problem.
- *It uses expensive algebra.* The approach of [33] uses pairing-based SNARKs instantiated via *pairing-friendly cycles of elliptic curves*. Only a *single* cycle construction is known, *MNT cycles*; it consists of two prime-order elliptic curves, with embedding degrees 4 and 6 respectively. Curves in an MNT cycle must be much bigger than usual in order to compensate for the loss of security caused by the small embedding degrees. Moreover the fields that arise from MNT cycles are *imposed on applications* rather than being chosen depending on the needs of applications, causing additional performance overheads. Attempts to find "better" cycles, without these limitations, have resulted in some negative results [64]. Indeed, finding *any other* cycles beyond MNT cycles is a challenging open problem.

5.1.1 Our results

We present a new methodology for recursive composition that simultaneously overcomes all of the limitations discussed above. We experimentally validate our methodology, demonstrating feasibility in practice.

The starting point of our work is the observation that recursive composition is simpler when applied to a SNARG (of knowledge) that supports *preprocessing*, as we explain in Section 5.2.1. This property of a SNARG means that in an offline phase one can produce a short summary for a given circuit and then, in an online phase, one may use this short summary to verify SNARGs that attest to the satisfiability of the circuit with different partial assignments to its inputs. The online phase can be as fast as reading the SNARG (and the partial assignment), and in particular sublinear in the circuit size *even for arbitrary circuits*. Throughout, by "preprocessing SNARG" we mean a SNARG whose verifier runs in time *polylogarithmic* in the circuit size.[2]

Our methodology has three parts: (1) a transformation that maps any "holographic proof" into a preprocessing SNARG in the random oracle model; (2) a holographic proof for (rank-1) constraint systems, which leads to a corresponding preprocessing SNARG; (3) a transformation that recurses any preprocessing SNARK (once the random oracle is heuristically instantiated via a cryptographic hash function).

We now summarize our contributions for each of these parts.

[2]In contrast, *non*-preprocessing SNARGs can achieve fast verification *only for structured circuits*, because the verification procedure must at a minimum read the *description* of the circuit whose satisfiability it checks. The description of a circuit can be much smaller than the circuit itself only when the circuit has suitable structure, e.g., repeated sub-components in parallel or in series.

(1) From holographic proofs to preprocessing SNARGs. A probabilistic proof is *holographic* if the verifier does not receive the circuit description as an input but, rather, makes a small number of queries to an encoding of the circuit [16]. Recent work [65] has established a connection between holography and preprocessing (which we review in Section 5.1.2). The theorem below adds to this connection, by showing that interactive oracle proofs (IOPs) [32, 127] that are holographic can be compiled into preprocessing SNARGs that are secure in the quantum random oracle model [48, 67].

Theorem 1 (informal)**.** *There is an efficient transformation that compiles any holographic IOP for a relation $\mathcal{R}$ into a preprocessing SNARG for $\mathcal{R}$ that is unconditionally secure in the random oracle model. If the IOP is a (honest-verifier) zero knowledge proof of knowledge then the transformation produces a zero knowledge SNARG of knowledge (zkSNARK). This extends to hold in the quantum random oracle model.*

By applying Theorem 1 to known holographic proofs for non-deterministic computations (such as the PCP in [16] or the IPCP in [90]), we obtain the first transparent preprocessing SNARG and the first post-quantum preprocessing SNARG. Unfortunately, known holographic proofs are too expensive for practical use, because encoding the circuit is costly (as explained in Section 5.1.2.1). In this paper we address this problem by constructing an efficient holographic proof, discussed below.

We note that holographic proofs involve relations $\mathcal{R}$ that consist of *triples* rather than *pairs* because the statement being checked has two parts. One part is called the *index*, which is encoded in an offline phase by the *indexer* and this encoding is provided as an oracle to the verifier. The other part is called the *instance*, which is provided as an explicit input to the verifier. For example, the index may be a circuit description and the instance a partial assignment to its inputs. We refer to this notion as *indexed relations* (see Section 5.3.2).

(2) Efficient protocols for R1CS. We present a holographic IOP for rank-1 constraint satisfiability (R1CS), a standard generalization of arithmetic circuits where the "circuit description" is given by coefficient matrices. We describe the corresponding indexed relation.

Definition 2 (informal)**.** *The indexed relation $\mathcal{R}_{\mathrm{R1CS}}$ is the set of triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, n, m, A, B, C), v, w\big)$$

where $\mathbb{F}$ is a finite field, A, B, C are $n \times n$ matrices over $\mathbb{F}$, each containing at most m non-zero entries, and $z := (v, w)$ is a vector in $\mathbb{F}^n$ such that $Az \circ Bz = Cz$. (Here "$\circ$" denotes the entry-wise product.)

Theorem 3 (informal)**.** *There exists a public-coin holographic IOP for the indexed relation $\mathcal{R}_{\mathrm{R1CS}}$ that is a zero knowledge proof of knowledge with the following efficiency features. In the offline phase, the encoding of an index is computable in $O(m \log m)$ field operations and consists of $O(m)$ field elements. In the online phase, the protocol has $O(\log m)$ rounds, with the prover using $O(m \log m)$ field operations and the verifier using $O(|v| + \log m)$ field operations. Proof length is $O(m)$ field elements and query complexity is $O(\log m)$.*

The above theorem improves, in the holographic setting, on prior IOPs for R1CS (see Fig. 5.1): it offers an exponential improvement in verification time compared to the linear-time verification in Chapter 3, and it offers succinct verification for all coefficient matrices compared to only structured ones as in Chapter 4.

Armed with an efficient holographic IOP, we use our compiler to construct an efficient preprocessing SNARG in the random oracle model. The following theorem is obtained by applying Theorem 1 to Theorem 3.

Theorem 4 (informal)**.** *There exists a preprocessing zkSNARK for R1CS that is unconditionally secure in the random oracle model (and the quantum random oracle model) with the following efficiency features. In the offline phase, anyone can publicly preprocess an index in time $O_\lambda(m \log m)$, obtaining a corresponding verification key of size $O_\lambda(1)$. In the online phase, the SNARG prover runs in time $O_\lambda(m \log m)$ and the SNARG verifier runs in time $O_\lambda(|v| + \log^2 m)$; argument size is $O_\lambda(\log^2 m)$.*

We have implemented the protocol underlying Theorem 4, obtaining the first efficient realization of a post-quantum transparent preprocessing zkSNARK.

For example, for a security level of 128 bits over a 181-bit prime field, arguments range from $80\,\mathrm{kB}$ to $160\,\mathrm{kB}$ for instances of up to millions of constraints. These argument sizes are two orders of magnitude bigger than *pre*-quantum *non*-transparent preprocessing zkSNARKs (see Section 5.1.2.2), and are $2\times$ bigger that the state of the art in post-quantum transparent *non*-preprocessing zkSNARKs (see Chapter 3). Our proving and verification times are comparable to prior work: proving takes several minutes, while verification takes several milliseconds *regardless of the constraint system*. (See Section 5.13.1 for performance details.)

Besides its application to post-quantum transparent recursion, our preprocessing zkSNARK provides attractive benefits over prior constructions, as we discuss in Section 5.1.2.2.

Note that, when the random oracle in the construction is heuristically instantiated via an efficient cryptographic hash function (as in our implementation), the resulting preprocessing zkSNARK is in the uniform reference string (URS) model, which means that the system parameters consist of a uniformly random string of fixed size.[3] The term "transparent" refers to a construction in the URS model.

(3) Post-quantum transparent recursion. We obtain the first efficient realization of post-quantum transparent recursive composition for SNARKs. The cryptographic primitive that formally captures this capability is known as *proof carrying data* (PCD) [68, 45], and so this is what we construct.

Theorem 5 (informal)**.** *There is an efficient transformation that compiles any preprocessing SNARK in the URS model into a preprocessing PCD scheme in the URS model. Moreover, if the preprocessing SNARK is post-quantum secure then so is the preprocessing PCD scheme.*

The above transformation, which preserves the "transparent" property and post-quantum security, is where recursive composition occurs. For details, including the notion of PCD, see Section 5.11.

[3] We stress that this step is a heuristic due to well-known limitations to the random oracle methodology [61, 89]. Investigating how to provably instantiate the random oracle for many natural constructions is an active research frontier.

Moreover, we provide an efficient implementation of the transformation in Theorem 5 applied to our implementation of the preprocessing zkSNARK from Theorem 4. The main challenge is to express the SNARK verifier's computation in as few constraints as possible, and in particular to design a constraint system for the SNARK verifier that on relatively small instances is smaller than the constraint system that it checks (thereby permitting arbitrary recursion depth). Via a combination of computer-assisted design and recent advances in algebraic hash functions, we achieve this threshold for all computations of at least 2 million constraints. Specifically, we can express a SNARK verifier checking 2 million constraints using only 1.1 million constraints, and this gap grows quickly with the computation size. *This is the first demonstration of post-quantum transparent recursive composition in practice.* (See Section 5.13.2 for performance details.)

	R1CS instances	holographic?	indexer time	prover time	verifier time	round complexity	proof length	query complexity
Chapter 3	arbitrary	NO	N/A	$O(m + n \log n)$	$O(\|\mathbb{x}\| + m)$	$O(\log n)$	$O(n)$	$O(\log n)$
Chapter 4 †	semi-succinct	NO	N/A	$O(m + n \log n)$	$O(\|\mathbb{x}\| + \log n)$	$O(\log n)$	$O(n)$	$O(\log n)$
this work	arbitrary	YES	$O(m \log m)$	$O(m \log m)$	$O(\|\mathbb{x}\| + \log m)$	$O(\log m)$	$O(m)$	$O(\log m)$

Figure 5.1: Comparison of IOPs for R1CS: two prior non-holographic IOPs, and our holographic IOP. Here n denotes the number of variables and m the number of non-zero coefficients in the matrices.
†: The parameters stated for Chapter 4 reflect replacing the constant-query low-degree test in the construction with a concretely-efficient logarithmic-query low-degree test such as [22], to simplify comparison.

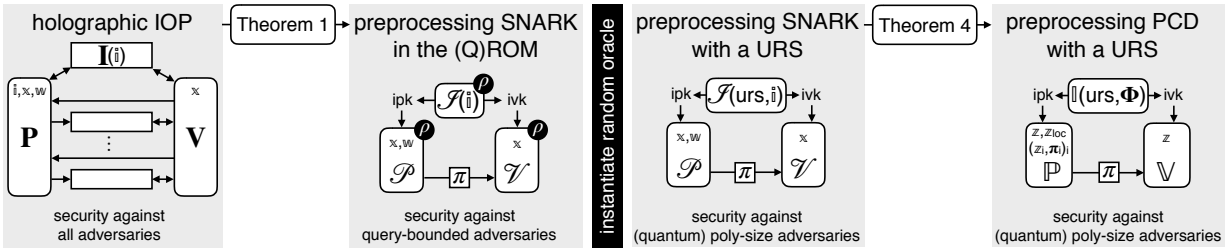


Figure 5.2: Diagram of our methodology for recursive composition that is post-quantum and transparent.

5.1.2 Comparison with prior work

We provide a comparison with prior work in the three areas to which we contribute: holographic proofs (Section 5.1.2.1); preprocessing SNARGs (Section 5.1.2.2); and recursive composition of SNARKs (Section 5.1.2.3). We omit a general discussion of the now ample literature on SNARGs, and in particular do not discuss *non*-preprocessing SNARGs for structured computations (e.g., [151], [23], and many others).

5.1.2.1 Prior holographic proofs

The verifier in a proof system cannot run in time that is sublinear in its input, because it must at a minimum read the input in order to know the statement being checked. Holographic proofs [16] avoid this limitation by considering a setting where the verifier does not receive its input explicitly but, instead, has query access to an encoding of it. The goal is then to verify the statement in time *sublinear* in its size; note that such algorithms are necessarily probabilistic.[4]

In Fig. 5.3 we compare the efficiency of prior holographic proofs and our holographic proof for the case of circuit satisfiability, where the input to the verifier is the description of an arbitrary circuit. There are two main prior holographic proofs in the literature. One is the PCP construction in [16], where it suffices for the verifier to query a few locations of a low-degree extension of the circuit description. Another one is the "bare bones" protocol in [90], which is a holographic IP for circuit *evaluation* that can be re-cast as a holographic IPCP for circuit *satisfaction*; the verifier relies on the low-degree extensions of functions that describe each layer of the circuit. The constructions in [16] and [90] are unfit for practical use as holographic proofs in Theorem 1, because encoding the circuit incurs a polynomial blowup due to the use of *multivariate* low-degree extensions (which yield encodings with inverse polynomial rate).

In the table we exclude the "algebraic holographic proof" of Marlin [65], because the soundness guarantee of such a proof is incompatible with Theorem 1.

Comparison with this work. Our holographic proof is the first to achieve efficient asymptotics not only for the prover and verifier, but also for the indexer, which is responsible for producing the encoding of the circuit.

	proof type	indexer time	prover time	verifier time
[16]	PCP	$\mathrm{poly}(N)$	$\mathrm{poly}(N)$	$\mathrm{poly}(\lvert\mathbb{x}\rvert + \log(N))$
[90]	IPCP	$\mathrm{poly}(N)$	$\mathrm{poly}(\lvert\mathbb{w}\rvert) + O(N)$	$O(\lvert\mathbb{x}\rvert + D \log W)$
this work	IOP	$O(N \log N)$	$O(N \log N)$	$O(\lvert\mathbb{x}\rvert + \log N)$

Figure 5.3: Comparison of holographic proofs for arithmetic circuit satisfiability. Here $\mathbb{x}$ denotes the known inputs, $\mathbb{w}$ the unknown inputs, and N the total number of gates; if the circuit is layered, D denotes circuit depth and W circuit width. Our Theorem 1 can be used to compile any of these holographic proofs into a preprocessing SNARG. (For better comparison with other works, [90] is stated as an IPCP for circuit satisfiability rather than as an IP for circuit evaluation; in the latter case, the prover time would be $O(N)$. The prover times for [90] incorporate the techniques for linear-time sumcheck and others introduced in [141, 151].)

[4] The goal of sublinear verification via holographic proofs is similar to, but distinct from, the goal of sublinear verification via *proximity proofs* (as, e.g., studied in [81, 78, 37, 130, 99]). In this latter setting, the verifier has oracle access to an input that is *not* promised to be encoded and, in particular, cannot in general decide if the input is in the language without reading all of the input. To allow for sublinear verification without any promises on the input, the decision problem is relaxed: the verifier is only asked to decide if the input is in the language or *far* from any input in the language.

5.1.2.2 Prior preprocessing SNARGs

Prior works construct preprocessing SNARGs in a model where a trusted party samples, in a parameter setup phase, a structured reference string (SRS) that is proportional to circuit size. We summarize the main features of these constructions, distinguishing between the case of circuit-specific SRS and universal SRS.

- *Circuit-specific SRS:* a circuit is given as input to the setup algorithm, which samples a (long) proving key and a (short) verification key that can be used to produce and validate arguments for the circuit. Preprocessing SNARGs with circuit-specific SRS originate in [94, 112, 87, 46], and have been studied in an influential line of work that has led to highly-efficient constructions (e.g., [95]) and large-scale deployments (e.g., [80]). They are obtained by combining linear interactive proofs and linear-only encodings. The argument sizes achievable in this setting are very small: less than 200 bytes.

- *Universal SRS:* a size bound is given as input to the setup algorithm, which samples a (long) proving key and a (short) verification key that can be used to produce and validate arguments for circuits within this bound. A public procedure can then be used to specialize both keys for arguments relative to the desired circuit. Preprocessing SNARGs with universal (and updatable) SRS were introduced in [96], and led to efficient constructions in [116, 65, 85]. They are obtained by combining "algebraic" holographic proofs (see below) and polynomial commitment schemes. The argument sizes currently achievable with universal SRS are bigger than with circuit-specific SRS: less than 1000 bytes.

Comparison with this work. Theorem 1 provides a methodology to obtain preprocessing SNARGs in the (quantum) random oracle model, which heuristically implies (by suitably instantiating the random oracle) preprocessing SNARGs that are post-quantum and transparent. Neither of these properties is achieved by prior preprocessing SNARGs. Theorem 1 also develops the connection between holography and preprocessing discovered in [65], which considers the case of holographic proofs where the completeness and soundness properties are restricted to "algebraic provers" (which output polynomials of prescribed degrees). We consider the case of general holographic proofs, where completeness and soundness are not restricted.

Moreover, our holographic proof (Theorem 3) leads to a preprocessing SNARG (Theorem 4) that, as supported by our implementation, provides attractive benefits over prior preprocessing SNARGs.

- Prior preprocessing SNARGs require cryptographic ceremonies to securely sample the long SRS, which makes deployments difficult and expensive. This has restricted the use of preprocessing SNARGs to proving relatively small computations, due to the prohibitive cost of securely sampling SRSs for large computations. This is unfortunate because preprocessing SNARGs could be useful for "scalability applications", which leverage succinct verification to efficiently check large computations (e.g., verifying the correctness of large batches of trades executed at a non-custodial exchange [18, 140]).

  The transparent property of our preprocessing SNARG means that the long SRS is replaced with a fixed-size URS (uniform reference string). This simplifies deployments and enables scalability applications.

- Prior preprocessing SNARGs are limited to express computations over the prime fields that arise as the scalar fields of pairing-friendly elliptic curves. Such fields are imposed by parametrized curve families that offer little flexibility for optimizations or applications. (Alternatively one can use the Cocks–Pinch method [84] to construct an elliptic curve with a desired scalar field, but the resulting curve is inefficient.)

  In contrast, our preprocessing SNARG is easily configurable across a range of security levels, and supports most large prime fields and all large binary fields, which offers greater flexibility in terms of performance optimizations and customization for applications.

Remark 5.1.1 (weaker forms of preprocessing)**.** Prior work proved recursive composition only for non-interactive arguments of knowledge with succinct verifiers [45]; this is the case for our definition of preprocessing SNARGs. In this paper we show that recursive composition is possible even when the verifier is merely *sublinear* in the circuit size (see Section 5.11), though the cost of each recursion is much steeper than in the polylogarithmic case. This provides additional motivation to the study of preprocessing with sublinear verifiers (e.g., [51, 135]).

5.1.2.3 Recursion for pairing-based SNARKs

The approach to recursive composition of [33] uses pairing-based (preprocessing) SNARKs based on pairing-friendly cycles of elliptic curves. This approach applies to constructions with circuit-specific SRS (e.g. [95]) *and* to those with universal SRS (e.g. [96, 116, 65, 85]).

Informally, pairing-based SNARKs support languages that involve the satisfiability of constraint systems over a field that is *different* from the field used to compute the SNARK verifier — this restriction arises from the mathematics of the underlying pairing-friendly elliptic curve used to instantiate the pairing. This seemingly mundane fact has the regrettable consequence that expressing the SNARK verifier's computation in the language supported by the SNARK (to realize recursive composition) is unreasonably expensive due to this "field mismatch". To circumvent this barrier, prior work leveraged *two* pairing-based SNARKs where the field to compute one SNARK verifier equals the field of the language supported by the other SNARK, and vice versa. This condition enables each SNARK to efficiently verify the other SNARK's proofs.

These special SNARKs rely on pairing-friendly cycles of elliptic curves, which are pairs of pairing-friendly elliptic curves where the base field of one curve equals the scalar field of the other curve and vice versa. The only known construction is *MNT cycles*, which consist of two prime-order elliptic curves with embedding degrees 4 and 6 respectively. An MNT cycle must be much bigger than usual in order to compensate for the low security caused by the small embedding degrees. For example, for a security level of 128 bits, curves in an MNT cycle must be defined over a prime field with roughly 800 bits; this is over *three times* the 256 bits that suffice for curves with larger embedding degrees. These performance overheads can be significant in practice, e.g., Coda [121] is a project that has deployed MNT cycles in a product, and has organized a community challenge to speed up the proof generation for pairing-based SNARKs [71]. A natural approach to mitigate this problem would be to find "high-security" cycles (i.e., with higher embedding degrees) but to date little is known about pairing-friendly cycles beyond a few negative results [64].

Comparison with this work. The approach to recursion that we present in this paper is *not tied* to constructions of pairing-friendly cycles of elliptic curves. In particular, our approach scales gracefully across different security levels, and also offers more flexibility when choosing the desired field for an application. In addition, our approach is post-quantum and, moreover, uses a transparent (i.e., public-coin) setup.

On the other hand, our approach has two disadvantages. First, argument size is about 100 times bigger than the argument size achievable by cycle-based recursion. Second, the number of constraints needed to express the verifier's computation is about 40 times bigger than those needed in the case of cycle-based recursion (e.g., the verifier of [95] can be expressed in about 40,000 constraints). The vast majority of these constraints come from the many hash function invocations required to verify the argument.

Both of the above limitations are somewhat orthogonal to our approach and arguably temporary: the large proof size and many hash invocations come from the many queries required from current constructions of low-degree tests [22, 35]. As the state of the art in low-degree testing progresses (e.g., to high-soundness constructions over large alphabets), both argument size and verifier size will also improve.

5.2 Techniques

We discuss the main ideas behind our results. In Section 5.2.1 we explain how preprocessing simplifies recursive composition. In Section 5.2.2 we describe our compiler from holographic IOPs to preprocessing SNARGs (Theorem 1). In Section 5.2.3 we describe our efficient holographic IOP (Theorem 3), and then in Section 5.2.4 we discuss the corresponding preprocessing SNARG (Theorem 4). In Section 5.2.5 we describe how to obtain post-quantum and transparent PCD (Theorem 5). In Section 5.2.6 we discuss our verifier circuit.

Recall that indexed relations consist of *triples* $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ is the instance, and $\mathbb{w}$ is the witness (see Section 5.3.2). We use these relations because the statements being checked have two parts, the index $\mathbb{i}$ (e.g., a circuit description) given in an offline phase and the instance $\mathbb{x}$ (e.g., a partial input assignment) given in an online phase.

5.2.1 The role of preprocessing SNARKs in recursive composition

We explain why preprocessing simplifies recursive composition of SNARKs. For concreteness we consider the problem of incrementally proving the iterated application of a circuit $F\colon \{0,1\}^n \to \{0,1\}^n$ to an initial input $z_0 \in \{0,1\}^n$. We are thus interested in proving statements of the form "*given z_T there exists z_0 such that $z_T = F^T(z_0)$*", but wish to avoid having the SNARK prover check the correctness of all T invocations at once. Instead, we break the desired statement into T smaller statements $\{\text{“}z_i = F(z_{i-1})\text{”}\}_{i=1}^{T}$ and then inductively prove them. Informally, for $i = 1, \ldots, T$, we produce a SNARK proof π_i for this statement:

> "*Given a counter i and claimed output z_i, there exists a prior output z_{i-1} such that $z_i = F(z_{i-1})$ and, if $i > 1$, there exists a SNARK proof π_{i-1} that attests to the correctness of z_{i-1}.*"

Formalizing this idea requires care, and in particular depends on how the SNARK achieves succinct verification (a prerequisite for recursive composition). There are two methods to achieve succinct verification.

(1) *Non-preprocessing SNARKs for structured computations.* The SNARK supports non-deterministic computations expressed as programs, i.e., it can be used to prove/verify statements of the form "given a program M, primary input $\mathbb{x}$, and time bound t, there exists an auxiliary input $\mathbb{w}$ such that M accepts $(\mathbb{x}, \mathbb{w})$ in t steps". (More generally, the SNARK could support any computation model for which the description of a computation can be significantly smaller than the size of the described computation.)

(2) *Preprocessing SNARKs for arbitrary computations.* The SNARK supports circuit satisfiability, i.e., it can be used to prove/verify statements of the form "given a circuit C and primary input $\mathbb{x}$, there exists an auxiliary input $\mathbb{w}$ such that $C(\mathbb{x}, \mathbb{w}) = 0$". Preprocessing enables the circuit C to be summarized into a short verification key ivk_C that can be used for succinct verification *regardless* of the structure of C. (More generally, the SNARK could support any computation model as long as preprocessing is possible.)

We compare the costs of recursive composition in these two cases, showing why the preprocessing case is cheaper. Throughout we consider SNARKs in the uniform reference string model, i.e., parameter setup consists of sampling a fully random string urs of size $\mathrm{poly}(\lambda)$ that suffices for proving/verifying any statement.

(1) Recursion without preprocessing. Let $(\mathcal{P}, \mathcal{V})$ be a *non*-preprocessing SNARK for non-deterministic program computations. In this case we follow [45]: recursion is realized via a program R, which depends on urs and F, that checks one invocation of the circuit F and the validity of a prior SNARK proof relative to the reference string urs. The program R is defined as follows:

> **Primary input:** a tuple $\mathbb{x} = (M, i, z_i)$ consisting of the description of a program M, counter i, and claimed output z_i. (We later set $M := R$ to achieve recursion, as explained shortly.)
>
> **Auxiliary input:** a tuple $\mathbb{w} = (z_{i-1}, \pi_{i-1})$ consisting of a previous output z_{i-1} and corresponding SNARK proof π_{i-1} that attests to its correctness.
>
> **Code:** $R(\mathbb{x}, \mathbb{w})$ accepts if $z_i = F(z_{i-1})$ and, if $i > 1$, $\mathcal{V}(\mathsf{urs}, M, \mathbb{x}_{i-1}, t, \pi_{i-1}) = 1$ where $\mathbb{x}_{i-1} := (M, i-1, z_{i-1})$ and t is a suitably chosen time bound.

The program R can be used to incrementally prove the iterated application of the circuit F. Given a tuple $(i-1, z_{i-1}, \pi_{i-1})$ consisting of the current counter, output, and proof, one can use the SNARK prover to obtain the next tuple (i, z_i, π_i) by setting $z_i := F(z_{i-1})$ and computing the proof $\pi_i := \mathcal{P}(\mathsf{urs}, R, \mathbb{x}_i, t, \mathbb{w}_i)$ for the instance $\mathbb{x}_i := (R, i, z_i)$ and witness $\mathbb{w}_i := (z_{i-1}, \pi_{i-1})$ (and a certain time bound t). Note that we have set $M := R$, so that (the description of) R is part of the primary input to R. A tuple (i, z_i, π_i) can then be verified by running the SNARK verifier, as $\mathcal{V}(\mathsf{urs}, R, \mathbb{x}_i, t, \pi_i)$ for $\mathbb{x}_i := (R, i, z_i)$.[5]

[5]The astute reader may notice that we could have applied the Recursion Theorem to the program R to obtain a new program R^* that has access to its own code, and thereby simplify primary inputs from triples $\mathbb{x} = (M, i, z_i)$ to pairs $\mathbb{x} = (i, z_i)$. This, however, adds unnecessary complexity. Indeed, here we can rely on the SNARK verifier to

We refer the reader to [45] for details on how to prove the above construction secure. The aspect that we are interested to raise here is that the program R is tasked to simulate itself, essentially working as a universal machine. This means that every elementary operation of R, and in particular of F, needs to be simulated by R in its execution. This essentially means that the computation time of R, which dictates the cost of each proof composition, is at least a constant $c > 1$ times the size of $|F|$. *This multiplicative overhead on the size of the circuit F, while asymptotically irrelevant, is a significant overhead in concrete efficiency*.

(2) Recursion with preprocessing. We describe how to leverage preprocessing in order to avoid universal simulation, and in particular to avoid *any* multiplicative performance overheads in recursive composition. Intuitively, preprocessing provides a "cryptographic simplification" to the requisite recursion, by enabling us to replace the description of the computation with a succinct cryptographic commitment to it.

Let $(\mathcal{I}, \mathcal{P}, \mathcal{V})$ be a preprocessing SNARK for circuits. Recursion is realized via a circuit R that depends on urs and F, and checks one invocation of F and a prior proof. The circuit R is defined as follows:

> **Primary input:** a tuple $\mathbb{x} = (\mathsf{ivk}, i, z_i)$ consisting of an index verification key ivk, counter i, and claimed output z_i. (We later set $\mathsf{ivk} := \mathsf{ivk}_R$ to achieve recursion.)
>
> **Auxiliary input:** a tuple $\mathbb{w} = (z_{i-1}, \pi_{i-1})$ consisting of a previous output z_{i-1} and corresponding SNARK proof π_{i-1} that attests to its correctness.
>
> **Code:** $R(\mathbb{x}, \mathbb{w})$ accepts if $z_i = F(z_{i-1})$ and, if $i > 1$, $\mathcal{V}(\mathsf{urs}, \mathsf{ivk}, \mathbb{x}_{i-1}, \pi_{i-1}) = 1$ where $\mathbb{x}_{i-1} := (\mathsf{ivk}, i-1, z_{i-1})$.

The circuit R can be used for recursive composition as follows. In the offline phase, we run the indexer $\mathcal{I}$ on the circuit R, obtaining a long index proving key ipk_R and a short index verification key ivk_R that can be used to produce and validate SNARKs with respect to the circuit R. Subsequently, in the online phase, one can use the prover $\mathcal{P}$ to go from a tuple $(i-1, z_{i-1}, \pi_{i-1})$ to a new tuple (i, z_i, π_i) by letting $z_i := F(z_{i-1})$ and $\pi_i := \mathcal{P}(\mathsf{urs}, \mathsf{ipk}_R, \mathbb{x}_i, \mathbb{w}_i)$ for the instance $\mathbb{x}_i := (\mathsf{ivk}_R, i, z_i)$ and witness $\mathbb{w}_i := (z_{i-1}, \pi_{i-1})$. Note that we have set $\mathsf{ivk} := \mathsf{ivk}_R$, so that the verification key ivk_R is part of the primary input to the circuit R. A tuple (i, z_i, π_i) can then be verified by running the SNARK verifier, as $\mathcal{V}(\mathsf{urs}, \mathsf{ivk}_R, \mathbb{x}_i, \pi_i)$ for $\mathbb{x}_i := (\mathsf{ivk}_R, i, z_i)$.

Crucially, the circuit R does *not* perform any universal simulation involving the circuit F, and in particular does not incur multiplicative overheads. Indeed, $|R| = |F| + |\mathcal{V}| = |F| + o(|F|)$. This was enabled by preprocessing, which let us provide the index verification key ivk_R as input to the circuit R.

In fact, preprocessing is *already* part of the efficient approach to recursive composition in [33]. There the preprocessing SNARK uses a structured, rather than uniform, reference string but the benefits of preprocessing are analogous (even when the reference string depends on the circuit or a bound on it).

In summary: preprocessing SNARKs play an important role in efficient recursive composition. Our first milestone is post-quantum and transparent preprocessing SNARKs, which we then use to achieve post-quantum and transparent recursive composition.

---

provide R with its own code as part of the primary input, obviating this extra step. (For reference, the Recursion Theorem states that for every program $A(x, y)$ there is a program $B(y)$ that computes $A(\langle B \rangle, y)$, where the angle brackets emphasize that the first argument is the description of the program B.)

5.2.2 From holographic proofs to preprocessing with random oracles

We describe the main ideas behind Theorem 1, which provides a transformation that compiles any holographic IOP for an indexed relation $\mathcal{R}$ into a corresponding preprocessing SNARG for $\mathcal{R}$. See Section 5.10 for details.

Warmup: holographic PCPs. We first consider the case of PCPs, a special case of IOPs. Recall that the Micali transformation [118] compiles a (non-holographic) PCP into a (non-preprocessing) SNARG. We modify this transformation to compile a *holographic* PCP into a *preprocessing* SNARG, by using the fact that the SNARG verifier output by the Micali transformation invokes the PCP verifier as a black box.

In more detail, the main feature of a holographic PCP is that the PCP verifier does not receive the index as an explicit input but, rather, makes a small number of queries to an encoding of the index given as an oracle. If we apply the Micali transformation to the holographic PCP, we obtain a SNARG verifier that must answer queries by the PCP verifier to the encoded index. If we simply provided the index as an input to the SNARG verifier, then we cannot achieve succinct verification and so would not obtain a preprocessing SNARG. Instead, we let the SNARG indexer compute the encoded index, compute a Merkle tree over it, and output the corresponding root as an *index verification key* for the SNARG verifier. We can then have the SNARG prover extend the SNARG proof with answers to queries to the encoded index, certified by authentication paths relative to the index verification key. In this way the SNARG verifier can use the answers in the SNARG proof to answer the queries to the encoded index by the underlying PCP verifier.

This straightforward modification to the Micali transformation works: one can prove that if the soundness error of the holographic PCP is ϵ then the soundness error of the preprocessing SNARG is $t\epsilon + O(t^2 \cdot 2^{-\lambda})$ against t-query adversaries in the random oracle model. (A similar expression holds for quantum adversaries.)

General case: holographic IOPs. While efficient constructions of holographic PCPs are not known, in this paper we show how to construct an efficient holographic IOP (see Section 5.2.3). Hence we are actually interested in compiling holographic IOPs. In this case our starting point is the BCS transformation [32], which compiles a (non-holographic) IOP into a (non-prepreprocessing) SNARG. We adopt a similar strategy as above: we modify the BCS transformation to compile a *holographic* IOP into a *preprocessing* SNARG, using the fact that the SNARG verifier output by the BCS transformation invokes the IOP verifier as a black box. Indeed, the main feature of a holographic IOP is the fact that the IOP verifier makes a small number of queries to an encoding of the index given as an oracle. Therefore the SNARG indexer can output the Merkle root of the encoded index as an index verification key, which subsequently the SNARG verifier can use to authenticate answers about the encoded index claimed by the SNARG prover.

An important technical difference here is the fact that the soundness error of the resulting preprocessing SNARG is not related to the soundness error of the holographic IOP but, instead, to its *state-restoration* (SR) soundness error, a stronger notion of soundness introduced in [32]. Namely, we prove that if the SR soundness error of the holographic PCP is $\epsilon_{\mathrm{sr}}(t)$ then the soundness error of the preprocessing SNARG is $\epsilon_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda})$. This phenomenon is inherited from the (unmodified) BCS transformation.

Proof of knowledge. All known constructions of PCD from SNARGs require that the SNARG

is an *argument of knowledge* (i.e., it is a SNARK). We show that if the holographic IOP is a (state-restoration) proof of knowledge (PoK), our transformation yields a preprocessing SNARK. In the simple non-adaptive setting this can be shown straightforwardly from known properties of the BCS transformation. However, in order to provide the strongest possible evidence for the existence of SNARKs satisfying the requirements for our IVC/PCD construction (see Section 5.2.5), we prove a stronger *adaptive* knowledge soundness property, which is not known to hold for the standard BCS transformation. We show that a standard modification of the BCS transformation achieves this stronger notion for a wide class of holographic IOPs.

Post-quantum security. A primary contribution of this work is to construct post-quantum preprocessing SNARKs (and to show that these yield post-quantum PCD). Using techniques developed in [152, 67], we show that our transformation yields a preprocessing SNARG in the QROM. Here the relevant soundness notion of the holographic IOP is the *round-by-round* (RBR) soundness error, defined in [60].

Moreover, if the holographic IOP is *round-by-round knowledge sound*, as defined in [67], our transformation yields a preprocessing SNARK in the QROM; this establishes, for the first time, the existence of adaptively-secure SNARKs in the QROM. We prove security by exhibiting a universal quantum extractor. As in the classical case, we prove a strong adaptive knowledge soundness property that is a close analogue for the property required in our IVC/PCD construction.

5.2.3 An efficient holographic proof for constraint systems

We describe the main ideas behind Theorem 3, which provides an efficient construction of a holographic IOP for rank-1 constraint satisfiability (R1CS). See Definition 2 for the indexed relation representing this problem.

Our starting point: Marlin. Our construction borrows ideas from the *algebraic holographic proof* (AHP) underlying Marlin, a pairing-based zkSNARK due to [65]. An AHP is similar to a holographic IOP, except that the indexer and the prover (both honest and malicious) send *low-degree univariate polynomials* rather than evaluations of functions. The verifier may evaluate these polynomials at any point in the field.

To understand how AHPs and holographic IOPs differ, it is instructive to consider how one might construct a holographic IOP from an AHP. A natural approach is to construct the indexer and prover for the hIOP as follows: run the indexer/prover of the AHP, and whenever the indexer/prover outputs a polynomial, evaluate it and send this evaluation as the oracle. There are several issues with this approach. First, hIOPs require a stronger soundness guarantee: soundness must hold against malicious provers that send *arbitrary* oracles. Second, evaluating the polynomial requires selecting a set $L \subseteq \mathbb{F}$ over which to evaluate it. In general, since the verifier in the AHP may query any point in $\mathbb{F}$, we would need to take $L := \mathbb{F}$, which is prohibitively expensive for the indexer and prover if $\mathbb{F}$ is much larger than the instance size (as it often is, for both soundness and application reasons). Third, assuming that one manages to decouple L and $\mathbb{F}$, the soundness error of one invocation of the AHP will (at best) decrease with $1/|L|$ instead of $1/|\mathbb{F}|$, which requires somehow reducing the soundness error of the AHP to, say, $1/2^{\lambda}$, and simply re-running in parallel the AHP for $\lambda - \log |L|$ would be expensive in all

relevant parameters.

The first issue could be resolved by composing the resulting protocol with a low-degree test. This introduces technicalities because we cannot hope to check that the oracle is exactly low-degree (as required in an AHP) — we can only check that the oracle is *close* to low-degree. The best way to resolve the second issue depends on the AHP itself, and would likely involve out-of-domain sampling [35]. Finally, resolving the third issue may not be possible in general (in fact, we do not see how resolve it for the AHP in Marlin).

These above issues show that, despite some similarities, there are markedly *different* design considerations on hIOPs versus AHPs. For this reason, while we will follow some of the ideas outlined above, we do not take the Marlin AHP as a black box. Instead, we will draw on the ideas underlying the Marlin AHP in order to build a suitable hIOP for this paper. Along the way, we also show how to reduce the round complexity of the Marlin AHP from 3 to 2, an ideas that we use to significantly improve the efficiency of our construction.

Aurora. The structure of our holographic IOP, like the Marlin AHP, follows the one of Aurora (Chapter 3), which we now briefly recall. Given an R1CS instance (A, B, C), the prover sends to the verifier f_z, the RS-encoding of a vector z, and three oracles f_A, f_B, f_C which are purportedly the RS-encodings of the three vectors Az, Bz, Cz respectively. The prover and verifier then engage in subprotocols to prove that (i) f_A, f_B, f_C are indeed encodings of Az, Bz, Cz, and (ii) $f_A \cdot f_B - f_C$ is an encoding of the zero vector.

Together these checks ensure that (A, B, C) is a satisfiable instance of R1CS. Testing (ii) is a straightforward application of known probabilistic checking techniques, and can be achieved with a logarithmic-time verifier. The primary challenge in the Aurora protocol (and protocols based on it) is testing (i).

In the Aurora protocol this is achieved via a reduction to univariate sumcheck, a univariate analogue of the sumcheck protocol in [115]. Univariate sumcheck also has a logarithmic verifier, but the reduction itself runs in time linear in the number of nonzero entries in the matrices A, B, C. A key technical contribution of the Marlin AHP is showing how to shift most of the cost of the reduction to the indexer in order to reduce the online cost of verification to logarithmic, as we now explain.

Challenges. We explain why the lincheck protocol of Section 3.5 is not holographic. Recall that the lincheck protocol, on input a matrix $M \in \mathbb{F}^{k \times k}$ and RS-encodings of vectors $\vec{x}, \vec{y} \in \mathbb{F}^k$, checks whether $\vec{x} = M\vec{y}$. It makes use of the following two facts: (i) for a vector of linearly-independent polynomials $\vec{u} \in \mathbb{F}[X]^k$ and any vectors $\vec{x}, \vec{y} \in \mathbb{F}^k$, if $\vec{x} \neq \vec{y}$ then the polynomials $\langle \vec{u}, \vec{x} \rangle$ and $\langle \vec{u}, \vec{y} \rangle$ are distinct, and so differ with high probability at a random $\alpha \in \mathbb{F}$, and (ii) for any matrix $M \in \mathbb{F}^{k \times k}$, $\langle \vec{u}, M\vec{y} \rangle = \langle \vec{u}M, \vec{y} \rangle$. The lincheck verifier sends a random $\alpha \in \mathbb{F}$ to the prover, and the prover then convinces the verifier that $\langle \vec{u}M, \vec{y} \rangle(\alpha) - \langle \vec{u}, \vec{x} \rangle(\alpha) = 0$ using the univariate sumcheck protocol.

This requires the verifier to evaluate the low-degree extensions of $\vec{u}_\alpha$ and $\vec{u}_\alpha M$ at a point $\beta \in \mathbb{F}$, where $\vec{u}_\alpha \in \mathbb{F}^k$ is obtained by evaluating each entry of $\vec{u}$ at α. This is equivalent to evaluating the bivariate polynomials $u(X, Y), u_M(X, Y) \in \mathbb{F}[X, Y]$, obtained respectively by extending $\vec{u}, \vec{u}M$ over Y, at a random point in $(\alpha, \beta) \in \mathbb{F}^2$. By choosing $\vec{u}$ appropriately, we can ensure that $u(X, Y)$ can be evaluated in logarithmic time (Section 4.6.1). But, without help from an indexer, evaluating $u_M(\alpha, \beta)$ requires time $\Omega(\|M\|)$.

A natural suggestion in the holographic setting is to have the indexer evaluate u_M over some domain $S \subseteq \mathbb{F} \times \mathbb{F}$, and make this evaluation part of the encoded index. This does achieve the goal of logarithmic verification time. Unfortunately, the degree of u_M in each variable is about k, and so even writing down the coefficients of u_M requires time $\Omega(k^2)$, which for sparse M is quadratic in $\|M\|$.

In the Marlin lincheck the indexer instead computes a certain *linear-size* (polynomial) encoding of M, which the verifier then uses in a multi-round protocol with the prover to evaluate u_M at its chosen point. Our holographic lincheck improves upon this protocol, reducing the number of rounds by one; we describe it next.

Our holographic lincheck. Recall from above that the lincheck verifier needs to check that $\langle \vec{u}, \vec{x} \rangle$ and $\langle \vec{u}M, \vec{y} \rangle$ are equal as polynomials in X. To do this, it will choose a random $\alpha \in \mathbb{F}$ and send it to the prover, then engage in the univariate sumcheck protocol to show that $\sum_h u(\alpha, h)\hat{x}(h) - u_M(\alpha, h)\hat{y}(h) = 0$, where $\hat{x}, \hat{y}$ are low-degree extensions of x and y.

To verify the above sum, the verifier must compute $u(\alpha, \beta)$ and $u_M(\alpha, \beta)$ for some $\beta \in \mathbb{F}$. The former can be computed by the verifier in logarithmic time as discussed; for the latter, we ask the prover to help. Specifically, we show that $u_M \equiv \hat{M^*}$, the unique bivariate low-degree extension of a matrix M^* which can be computed in quasilinear time from M (and in particular has $\|M^*\| = \|M\|$). Hence to show that $u_M(\alpha, \beta) = \gamma$ the prover and verifier can engage in a holographic *matrix arithmetization* protocol for M^* to show that $\hat{M^*}(\alpha, \beta) = \gamma$. Marlin makes use of a similar matrix arithmetization protocol, but for M itself, with a subprotocol to compute u_M from $\hat{M}$, which is a cost that we completely eliminate. Another improvement is that for our matrix arithmetization protocol we can efficiently reduce soundness error even when using a low-degree test, due to its non-recursive use of the sumcheck protocol.

Matrix arithmetization. Our matrix arithmetization protocol is a holographic IOP for computing the low-degree extension of a matrix $M \in \mathbb{F}^{H \times H}$ (provided in the index). It is useful here to view M in its sparse representation as a map $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ for some $K \subseteq \mathbb{F}$, where if $\langle M \rangle(k) = (a, b, \gamma)$ for some $k \in K$ then $M_{a,b} = \gamma$, and $M_{a,b} = 0$ otherwise.

The indexer computes $\hat{\mathsf{row}}, \hat{\mathsf{col}}, \hat{\mathsf{val}}$ which are the unique low-degree extensions of the functions $K \to \mathbb{F}$ induced by restricting $\langle M \rangle$ to its first, second, and third coordinates respectively, and outputs their evaluations over L. It is not hard to verify that

$$\hat{M}(\alpha, \beta) = \sum_{k \in K} L_{H,\hat{\mathsf{row}}(k)}(\alpha) L_{H,\hat{\mathsf{col}}(k)}(\beta) \hat{\mathsf{val}}(k) \ ,$$

for any $\alpha, \beta \in \mathbb{F}$, where $L_{H,a}$ is the polynomial of minimal degree which is 1 on a and 0 on $H \setminus \{a\}$. In order to check this equation using the sumcheck protocol we must modify the right-hand side: the summand must be a polynomial which can be efficiently evaluated. To this end, we make use of the "unnormalized Lagrange" polynomial $u_H(X, Y) := (v_H(X) - v_H(Y))/(X - Y)$ from [30]. This polynomial has the property that for every $a, b \in H$, $u_H(a, b)$ is 0 if $a \neq b$ and nonzero if $a = b$; and it is easy to evaluate at every point in $\mathbb{F}$. By having the indexer renormalize $\hat{\mathsf{val}}$ appropriately, we obtain

$$\hat{M}(X, Y) \equiv \sum_{k \in K} u_H(\hat{\mathsf{row}}(k), \alpha) u_H(\hat{\mathsf{col}}(k), \beta) \hat{\mathsf{val}}(k) \ .$$

We have made progress, but now the summand has quadratic degree: $\Omega(|H||K|)$ because we *compose* the polynomials u_H and $\hat{\mathsf{row}}, \hat{\mathsf{col}}$. Next we show how to avoid this composition.

Observe that since the image of K under $\hat{\mathsf{row}}, \hat{\mathsf{col}}$ is contained in H, $v_H(\hat{\mathsf{row}}(k)) = v_H(\hat{\mathsf{col}}(k)) = 0$. Hence the rational function

$$\frac{v_H(\alpha)}{(\alpha - \hat{\mathsf{row}}_{\langle M \rangle}(X))} \cdot \frac{v_H(\beta)}{(\beta - \hat{\mathsf{col}}_{\langle M \rangle}(X))} \cdot \hat{\mathsf{val}}_{\langle M \rangle}(X)$$

agrees with the summand on K; it is a rational extension of the summands. Moreover, the degrees of the numerator and denominator of the function are both $O(|K|)$. Now it remains to design a protocol to check the sum of a univariate rational function.

Rational sumcheck. Suppose that we want to check that $\sum_{k \in K} N(k)/D(k) = \gamma$, where N, D are low-degree polynomials. First, we have the prover send the (evaluation of the) unique polynomial f of degree $|K| - 1$ which agrees with N/D on K; that is, the unique low-degree extension of N/D viewed as a function from K to $\mathbb{F}$. We can use the *standard* univariate sumcheck protocol (Section 3.4) to test that $\sum_{k \in K} f(k) = \gamma$.

It then remains to check that f does indeed agree with N/D on K. This is achieved using standard techniques: if $N(k)/D(k) = f(k)$ for all $k \in K$, then $N(k) = D(k) \cdot f(k)$ for all $k \in K$ (at least if D does not vanish on K). Then $N - D \cdot f$ is a polynomial vanishing on K, and so is divisible by v_K. This can be checked using low-degree testing; for more details, see Section 5.5. Moreover, the degree of this equation is $\max(\deg(N), \deg(D) + |K|)$; in the matrix arithmetization protocol, this is $O(|K|)$.

Proof of knowledge and zero knowledge. Our full protocol for R1CS is a proof of knowledge, because when the verifier accepts with high enough probability it is possible to decode f_z into a satisfying assignment. We further achieve zero knowledge via techniques from Chapter 3. (Note that zero knowledge is not relevant for the matrix arithmetization protocol because the constraint matrices A, B, C are public.)

5.2.4 Post-quantum and transparent preprocessing

If we apply the compiler described in Section 5.2.2 (as captured in Theorem 1) to the efficient holographic proof for R1CS described in Section 5.2.3 (as captured in Theorem 3) then we obtain an efficient preprocessing zkSNARK for R1CS that is unconditionally secure in the (quantum) random oracle model (as captured in Theorem 4). We refer to the resulting construction as FRACTAL.

Implementation. We have implemented FRACTAL by extending the `libiop` library to support generic compilation of holographic proofs into preprocessing SNARGs, and then writing in code our holographic proof for R1CS. Our implementation supports a range of security levels and fields. (The only requirement on the field is that it contains certain smooth subgroups.) See Section 5.12.1 for more details on the implementation.

Clearly, the security of our implementation relies on the random oracle methodology applied to preprocessing SNARGs produced by our compiler, namely, we assume that if we replace every call to the random oracle with a call to a cryptographic hash function then the resulting

construction, which formally is in the URS model, inherits the relevant security properties that we proved in the (quantum) random oracle model.

Evaluation. We have evaluated FRACTAL, and its measured performance is consistent with asymptotic predictions. In particular, the polylogarithmic argument size and verification time quickly become smaller than native witness size and native execution time as the size of the checked computation increases.

We additionally compare the costs of FRACTAL to prior preprocessing SNARGs, finding that (a) our prover and verifier times are comparable to prior constructions; (b) argument sizes are larger than prior constructions (that have an SRS). The larger argument sizes of FRACTAL are nonetheless comparable with other post-quantum transparent *non*-preprocessing SNARGs. See Section 5.13.1 for more details on evaluation.

5.2.5 Post-quantum and transparent recursive composition

We summarize the ideas behind our contributions to recursive composition of SNARKs.

Proof-carrying data. Recursive composition is captured by a cryptographic primitive called *proof-carrying data* (PCD) [68, 45], which will be our goal. Consider a network of nodes, where each node receives messages from other nodes, performs some local computation, and sends the result on. PCD is a primitive that allows us to check the correctness of such distributed computations by recursively producing proofs of correctness for each message. Here "correctness" is locally specified by a *compliance predicate* Φ, which takes as input the messages received by a node and the message sent by that node (and possibly some auxiliary local data). A distributed computation is then considered Φ-*compliant* if, for each node, the predicate Φ accepts the node's messages (and auxiliary local data).

PCD captures proving the iterated application of a circuit as in Section 5.2.1, in which case the distributed computation evolves along a path. PCD also captures more complex topologies, which is useful for supporting distributed computations on long paths (via "depth-reduction" techniques [144, 45]) and for expressing dynamic distributed computations (such as MapReduce computations [69]).

From random oracle model to the URS model. While we have so far discussed constructions that are unconditionally secure in the (quantum) random oracle model, for recursion we now leave this model (by heuristically instantiating the random oracle with a cryptographic hash function) and start from preprocessing SNARKs in the URS model. The reason for this is far from mundane (and not motivated by implementation), as we now explain. The verifiers from Theorem 1 make calls to the random oracle, and therefore proving that the verifier has accepted would require using a SNARK that can prove the correctness of computations *in a relativized world where the oracle is a random function*. There is substantial evidence from complexity theory that such SNARKs do not exist (e.g., probabilistic proofs do not relativize with respect to a random oracle [63, 83, 66]). By instantiating the random oracle, all oracle calls can be "unrolled" into computations that do not involve oracle gates, and thus we can prove the the correctness of the resulting computation.[6] We stress that random oracles cannot be securely

[6]The necessity to instantiate the random oracle before recursion also arises in the first construction of incrementally verifiable computation [144]. One way to circumvent this difficulty is to consider oracles that are equipped

instantiated in the general case [61], and so we will assume that there is a secure instantiation of the random oracle for the preprocessing SNARKs produced via Theorem 1 (which, in particular, preserves proof of knowledge).

From SNARK to PCD. We prove that any preprocessing SNARK in the URS model can be transformed into a preprocessing PCD scheme in the URS model (Theorem 5.11.5).[7] The construction, described in Section 5.11, realizes recursive composition by following the template given in Section 5.2.1, except that the compliance predicate Φ may expect multiple input messages. This construction simplifies that of [45] for preprocessing SNARKs in the SRS model: we do not need to rely on collision-resistant hash functions to shrink the verification key ivk because we require it to be succinct, as captured in Lemma 5.11.8.[8]

Security against quantum adversaries. A key feature of our result (Theorem 5.11.5) is that we prove that if the SNARK is secure (i.e., is a proof of knowledge) against quantum adversaries then so is the resulting PCD scheme (i.e., it is also a proof of knowledge). Therefore, if we assume that FRACTAL achieves proof of knowledge against quantum adversaries when the random oracle is suitably instantiated, then by applying our result to FRACTAL we obtain a *post-quantum* preprocessing PCD scheme in the URS model.

For this result we require a suitable definition of adaptive proof of knowledge in the quantum setting. Our definition was chosen to achieve the following two goals: it must be strong enough to imply security for our PCD construction, and it must have a (Q)ROM analogue that is fulfilled by FRACTAL.

We highlight below two important issues that arose in finding this definition.

The proof of [45] uses the fact that, in the classical case, we may assume that the adversary is deterministic by selecting its randomness. This is not the case for quantum adversaries, as a quantum circuit can create its own randomness (e.g. by measuring a qubit in superposition). This means that we must be careful in defining the proof-of-knowledge property we require of the underlying SNARK. In particular, we must ensure that when we recursively extract proofs, these proofs are consistent with previously extracted proofs. For deterministic adversaries, this is implied by the proof of knowledge definition in [45]; for quantum adversaries, it is not.

A further complication arises when defining proof of knowledge in the QROM. In the classical (non-programmable) ROM, we can view the extractor and verifier as machines interacting with the same "real" oracle; in particular, the extractor simply passes the adversary's queries to the real oracle and notes the answers. Hence we can ask for an extraction guarantee of the type: "whenever the verifier accepts, the extractor succeeds" (this is the definition in [45]). In the QROM, no-cloning precludes this view: the extractor cannot act as an intermediary between the adversary and an external random oracle but must instead simulate the oracle itself.

Given these issues, we arrive at the following knowledge soundness definition. We observe

with a public verification procedure [68], however this requires embedding a secret in the oracle, which does not lend itself to straightforward software realizations and so we do not consider this approach in this paper.

[7] Analogously to a SNARK, here *preprocessing* denotes the fact that the PCD scheme enables succinct verification regardless of the computation expressed by the compliance predicate Φ (as opposed to only for structured computations).

[8] In contrast, the verification key ivk in [45] is allowed to grow linearly with the public input to the circuit that it summarizes, and so recursion required replacing ivk with a short hash of it, and moving ivk to the witness of the recursion circuit.

that the function of the strong extraction guarantee in [45] is to ensure closeness in distribution between the outputs of the prover and the extractor. (This is similar to the *witness-extended emulation* property introduced by [111].) To emulate this in the QROM, we simply impose the closeness-in-distribution requirement explicitly. In particular, we require that if the prover outputs some auxiliary information, the joint distribution of that auxiliary information and the adaptively-chosen instance is maintained in the output of the extractor. We show that this is both achievable in the QROM and sufficient for PCD in the standard model.

5.2.6 The verifier as a constraint system

In order to recursively compose FRACTAL (the preprocessing zkSNARK discussed in Section 5.2.4), we need to express FRACTAL's verifier as a constraint system. The size of this constraint system is crucial because this determines the threshold at which recursive composition becomes possible. Towards this goal, we design and implement a constraint system that applies to a general class of verifiers, as outlined below. FRACTAL's verifier is obtained as an instantiation within this class. See Section 5.12.2 for details.

Hash computations introduced by the compiler. Our compiler (Theorem 1) transforms any holographic IOP into a corresponding preprocessing SNARG, while preserving relevant zero knowledge or proof of knowledge properties. The preprocessing SNARG verifier makes a black-box use of the holographic IOP verifier, which means that we can design a *single* (parametrized) constraint system representing the transformation that works for *any* holographic IOP. All additional computations introduced by the compiler involve cryptographic hash functions (which heuristically instantiate the random oracle). In particular, there are two types of hash computations: (1) a hash chain computation used to derive the randomness for each round of the holographic IOP verifier, based on the Merkle roots provided by the preprocessing SNARG prover; and (2) verification of Merkle tree authentication paths in order to ensure the validity of the query answers provided by the preprocessing SNARG prover. We design generic constraint systems for both of these tasks. Since we are designing constraint systems it is more efficient to consider multiple hash functions specialized to work in different roles: a hash function to absorb inputs or squeeze outputs in the hash chain; a hash function to hash leaves of the Merkle tree; a many-to-one hash function for the internal nodes of the Merkle tree; and others.

Choice of hash function. While our implementation is generic with respect to the aforementioned hash functions (replacing any one of them with another is straightforward), the choice of hash function is nonetheless critical for concrete efficiency as we now explain. Expressing standard cryptographic hash functions, such as from the SHA or Blake family, as a constraint system requires more than 20,000 constraints. While this is acceptable for certain applications, these costs are prohibitive for hash-intensive computations, as is the case for the verifiers output by our compiler. Fortunately, the last few years have seen exciting progress in the design of *algebraic hash functions* [12, 4, 93, 7, 5], which by design can be expressed via a small number of arithmetic constraints over large finite fields. While this is an active research front, and in particular no standards have been agreed upon, many of the proposed functions are *significantly cheaper* than prior ones, and their security analyses are promising. In this work we decide to use one of these as our choice of hash function (Poseidon [93]). We do not claim that this is the "best"

choice among the currently proposed ones. (In fact, we know how to achieve better results via a combination of different choices.) We merely make one choice that we believe to be reasonable, and in particular suffices to demonstrate the feasibility of our methodology in practice.

Holographic IOP computations. The constraint system that represents the holographic IOP verifier will, naturally, depend on the specific protocol that is provided as input to the compiler.

That said, all known efficient IOPs, holographic or otherwise, are obtained as the combination of two ingredients: (1) a low-degree test for the Reed–Solomon (RS) code; and (2) an RS-encoded IOP, which is a protocol where the verifier outputs a set of algebraic claims, known as rational constraints, about the prover's messages. Examples of IOPs that fall in this category include our holographic IOP for R1CS, as well as protocols for R1CS in [9] and Chapters 3 and 4, and for AIR in [23].

We thus provide two constraint systems that target these two components. First, we provide a constraint system that realizes the FRI low-degree test [22], which is used in many efficient IOPs, including in our holographic IOP for R1CS. Second, we provide infrastructure to write constraint systems that express a desired RS-encoded IOP. This essentially entails specifying how many random elements the verifier should send in each round of the protocol, and then specifying constraints that express the rational constraints output by the verifier at the end of the RS-encoded IOP.

We then use the foregoing infrastructure to express the verifier of our holographic IOP for R1CS as a constraint system. We note that the very same generic infrastructure would make it straightforward to express the verifiers of other protocols with the same structure, such as those from Chapters 3 and 4 and [9, 23].

Remark 5.2.1 (succinct languages)**.** Our work in writing constraints for the verifier is restricted to non-uniform computation models such as R1CS (i.e., we are not concerned about the global structure of the constraint system). We do not claim to have an efficient way to express the same verifier via succinct languages such as AIR [23] or Succinct-R1CS (Definition 4.1.1). Doing so remains an open problem that, if addressed, may lead to additional opportunities in recursive composition (through *non*-preprocessing SNARKs).

5.3 Preliminaries

We state time costs in terms of basic operations over a given field $\mathbb{F}$, and size costs in terms of field elements in $\mathbb{F}$. We use the "big-oh" notation $O_{\mathbb{F}}$ to remind the reader that $\mathbb{F}$-operations and $\mathbb{F}$-elements have unit cost.

5.3.1 Sparse representations of matrices

Our protocols leverage sparse representations of matrices for efficiency, following the definition below. The definition is primarily for convenience in the sense that any reasonable sparse representation of a matrix can be transformed, in linear time, into one that follows the definition that we use.

Definition 5.3.1. *Let $H, K \subseteq \mathbb{F}$. A* **sparse representation** *of a matrix is a function $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ that is injective when its output is restricted to $H \times H$. The matrix $M \in \mathbb{F}^{H \times H}$ is obtained from $\langle M \rangle$ by setting, for $a, b \in H$, $M_{a,b} := \gamma$ if there exists $k \in K$ such that $\langle M \rangle(k) = (a, b, \gamma)$ and $M_{a,b} := 0$ otherwise.*

Note that a matrix $M \in \mathbb{F}^{H \times H}$ has many possible sparse representations. In particular, we may choose any large enough K and any injection from K to $H \times H$ that "covers" the nonzero entries of M.

5.3.2 Indexed relations

An *indexed relation* $\mathcal{R}$ is a set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ the instance, and $\mathbb{w}$ the witness; the corresponding *indexed language* $\mathcal{L}(\mathcal{R})$ is the set of pairs $(\mathbb{i}, \mathbb{x})$ for which there exists a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$. For example, the indexed relation of satisfiable boolean circuits consists of triples where $\mathbb{i}$ is the description of a boolean circuit, $\mathbb{x}$ is an assignment some of the input wires, and $\mathbb{w}$ is an assignment to the remaining input wires that makes the circuit output 0.

In this paper we build protocols for the indexed relation that represents *rank-1 constraint satisfiability* (R1CS), a generalization of arithmetic circuits where the "circuit description" is given by coefficient matrices. This is the same as Definition 3.6.1, but here we explicitly split the problem description into index and instance, and specify a sparse representation.

Definition 5.3.2 (R1CS indexed relation)**.** *The indexed relation $\mathcal{R}_{\mathrm{R1CS}}$ is the set of all triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, H, K, \langle A \rangle, \langle B \rangle, \langle C \rangle), (I, v), w\big)$$

where $\mathbb{F}$ is a finite field, H, K are subsets of $\mathbb{F}$, $\langle A \rangle, \langle B \rangle, \langle C \rangle \colon K \to H \times H \times \mathbb{F}$ are sparse representations of $H \times H$ matrices over $\mathbb{F}$, I is a subset of H, $v \in \mathbb{F}^I$, $w \in \mathbb{F}^{H \setminus I}$, and $z := (v, w) \in \mathbb{F}^H$ is a vector such that $Az \circ Bz = Cz$. (Here "$\circ$" denotes the entry-wise product between two vectors.)

Remark 5.3.3. The above definition can be generalized to the case where the matrices are *non-square*, namely, the matrices are in $\mathbb{F}^{H_1 \times H_2}$ for distinct domains $H_1, H_2 \subseteq \mathbb{F}$. All results stated in this chapter extend to this non-square case. Our focus on the square case is only for simplicity of exposition.

5.3.3 Algebra

The notational conventions in this chapter regarding the Reed–Solomon code are slightly different to the previous.

Reed–Solomon code. Given a subset L of a field $\mathbb{F}$ and degree bound $d < |L|$, we denote by $\mathrm{RS}\,[L, d] \subseteq \mathbb{F}^L$ all evaluations over L of univariate polynomials of degree at most d:

$$\mathrm{RS}\,[L, d] := \left\{ f \colon L \to \mathbb{F} \text{ s.t. } \exists\, \hat{f} \in \mathbb{F}[X] \text{ with } \deg(\hat{f}) \leq d \text{ and } \hat{f}(L) = f \right\} .$$

Whenever a polynomial $\hat{f}$ as above exists, then $\hat{f}$ is unique. The *rate* of $\mathrm{RS}[L,d]$ is $\rho := (d+1)/|L| \in (0,1)$, and its *distance* is $1-\rho$. The message encoded by $f \in \mathrm{RS}[L,d]$ is the restriction of $\hat{f}$ to a distinguished subset $H \subseteq \mathbb{F}$ of size $d+1$. (Note that H need not be a subset of L.) Observe that for every polynomial $\hat{f} \in \mathbb{F}[X]$ with degree less than $|L|$ it holds that the word $f_L := \hat{f}|_L$ is in $\mathrm{RS}\left[L, \deg(\hat{f})\right]$ (we will drop the subscript when the choice of domain L is clear from context). This means that there is a bijection between words in $\mathrm{RS}[L,d]$ and polynomials in $\mathbb{F}[X]$ of degree at most d.

Domains with subgroup structure. For a finite field $\mathbb{F}$, by "subgroup of $\mathbb{F}$" we mean either a subgroup of the additive group of $\mathbb{F}$ or a subgroup of $\mathbb{F}^*$. By "coset of $\mathbb{F}$" we mean a coset of a subgroup of $\mathbb{F}$ (additive or multiplicative). Throughout the paper we assume that the domain L for the Reed–Solomon code has "smooth" subgroup structure, meaning that it factors as a direct product of small (i.e., constant-size) subgroups. Under this assumption we can encode a message using the Reed–Solomon code in time $O_{\mathbb{F}}(|L| \log |L|)$. This assumption is also required by the low-degree test that we use [22, 35].

Vanishing polynomials. For a finite field $\mathbb{F}$ and subset $S \subseteq \mathbb{F}$, we denote by v_S the unique non-zero monic polynomial of degree at most $|S|$ that is zero on S; v_S is called the *vanishing polynomial* of S. If S is a coset in $\mathbb{F}$ then the coefficients of v_S can be found in time $O_{\mathbb{F}}(\log^2 |S|)$, and subsequently v_S can be evaluated at any point in time $O_{\mathbb{F}}(\log |S|)$.[9] In the holographic setting we can have the indexer find v_S for any S of interest, so that the verifier can evaluate v_S in time $O_{\mathbb{F}}(\log |S|)$. In this paper we assume that this is the case, so that for any coset S in $\mathbb{F}$ we can evaluate its vanishing polynomial at any point in time $O_{\mathbb{F}}(\log |S|)$.

Derivative of vanishing polynomials. We rely on various properties of the bivariate polynomial u_S related to the formal derivative of v_S, first exploited to obtain efficient probabilistic proofs in [30]. For a finite field $\mathbb{F}$ and subset $S \subseteq \mathbb{F}$, we define

$$u_S(X,Y) := \frac{v_S(X) - v_S(Y)}{X - Y} \ ,$$

which is a polynomial of individual degree $|S|-1$ because $X-Y$ divides $X^i - Y^i$ for any positive integer i. Note that $u_S(X,X)$ is the formal derivative of the vanishing polynomial $v_S(X)$.[10]

The bivariate polynomial $u_S(X,Y)$ satisfies two useful algebraic properties. First, it is strongly related to the Lagrange polynomials $L_{a,S}$ for $a \in S$. Specifically, $u_S(X,a) \equiv u_S(a,X) \equiv L_{a,S}(X) \cdot u_S(a,a)$ for all $a \in S$. In particular, this implies that the polynomials $(u_S(X,a))_{a \in S}$ are linearly independent. Second, the (unique) low-degree extension (in Y) of the vector $(u_S(X,a))_{a \in S} \in \mathbb{F}[X]^S$ is precisely $u_S(X,Y)$.

[9] If S is a multiplicative subgroup of $\mathbb{F}$ then $v_S(X) = X^{|S|} - 1$. More generally, if S is a ξ-coset of a multiplicative subgroup S_0 (namely, $S = \xi S_0$) then $v_S(X) = \xi^{|S|} v_{S_0}(X/\xi) = X^{|S|} - \xi^{|S|}$. In either case, v_S can be found and then evaluated at any point in time $O_{\mathbb{F}}(\log |S|)$. If instead S is an additive subgroup then there is an algorithm to find the coefficients of v_S in time $O_{\mathbb{F}}(\log^2 |S|)$. Being a linearized polynomial, v_S has only $O(\log |S|)$ nonzero coefficients, and in particular can be evaluated at any point in time $O_{\mathbb{F}}(\log |S|)$. An analogous statement holds if S is a coset of an additive subgroup.

[10] This follows from the general fact that, for $g(X,Y) := (f(X) - f(Y))/(X-Y)$, $g(X,X)$ is the formal derivative of $f(X)$. To see this, observe that $(X^n - Y^n)/(X-Y) = \sum_{i=0}^{n-1} X^i Y^{n-i-1}$. Setting $Y := X$ yields nX^{n-1}, the derivative of X^n.

If S is a coset in $\mathbb{F}$, an expression for $u_S(X, Y)$ can be found in time $O_{\mathbb{F}}(\log^2 |S|)$, and subsequently one can use this expression to evaluate $u_S(X, Y)$ at any point $(\alpha, \beta) \in \mathbb{F}^2$ in time $O_{\mathbb{F}}(\log |S|)$.[11]

Univariate sumcheck for cosets. For $S \subseteq \mathbb{F}, \hat{g} \in \mathbb{F}[X], \sigma \in \mathbb{F}$, define the polynomial:

$$\Sigma_S(\hat{g}, \sigma) := \begin{cases} X\hat{g}(X) + \sigma/|S| & \text{if } S \text{ is a multiplicative coset of } \mathbb{F} \\ \hat{g}(X) + \sigma X^{|S|-1} / \sum_{\alpha \in S} \alpha^{|S|-1} & \text{if } S \text{ is an additive coset of } \mathbb{F} \end{cases}.$$

Note that $\Sigma_S(\cdot\,, \cdot)$ may be viewed as an arithmetic circuit. We restate the univariate sumcheck lemma (Theorem 3.4.2) in a generic way using the above definition.

Lemma 5.3.4 (Univariate sumcheck)**.** *Let S be a coset of $\mathbb{F}$, and let $\hat{f} \in \mathbb{F}[X]$ be such that $\deg(\hat{f}) < |S|$. Then $\sum_{\alpha \in S} \hat{f}(\alpha) = \sigma$ if and only if there exists $\hat{g}$ with $\deg(\hat{g}) < |S| - 1$ such that $\hat{f} \equiv \Sigma_S(\hat{g}, \sigma)$.*

5.4 Definition of holographic IOPs

A **holographic IOP** for an indexed relation $\mathcal{R}$ is specified by a tuple $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$, where $\mathbf{I}$ is the *indexer*, $\mathbf{P}$ the *prover*, and $\mathbf{V}$ the *verifier*. The indexer is a deterministic polynomial-time algorithm, while the prover and verifier are probabilistic polynomial-time interactive algorithms. In an offline phase, given an index $\mathbb{i}$, the indexer $\mathbf{I}$ computes an encoding of $\mathbb{i}$, denoted $\mathbf{I}(\mathbb{i})$. Subsequently, in an online phase, the prover $\mathbf{P}$ receives as input a triple $(\mathbb{i}, \mathbb{x}, \mathbb{w})$, while the verifier $\mathbf{V}$ receives as input $\mathbb{x}$ and is granted oracle access to the encoded index $\mathbf{I}(\mathbb{i})$. The online phase consists of multiple rounds, and in each round the verifier $\mathbf{V}$ sends a message ρ_i and the prover $\mathbf{P}$ replies with a proof string Π_i, which the verifier can query at any location. At the end of the interaction, the verifier $\mathbf{V}$ accepts or rejects.

We say that HOL has perfect completeness and soundness error ϵ if the following holds.

- **Completeness.** For every index-instance-witness triple $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$, the probability that $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept in the interactive oracle protocol is 1.
- **Soundness.** For every index-instance pair $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ and prover $\tilde{\mathbf{P}}$, the probability that $\tilde{\mathbf{P}}$ convinces $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ to accept in the interactive oracle protocol is at most ϵ.

The *round complexity* k is the number of back-and-forth message exchanges between the verifier and the prover. The *proof length* L is the sum of the length of the encoded index plus the lengths of all proof strings sent by the prover. The *query complexity* q is the total number of queries made by the verifier; this includes queries to the encoded index and to the oracles sent by the prover.

The holographic IOPs that we construct achieve the stronger property of *knowledge soundness* and optionally also *zero knowledge*. We define both of these properties below.

[11] If S is a multiplicative coset in $\mathbb{F}$ then $u_S(X, Y) = (X^{|S|} - Y^{|S|})/(X - Y)$ and $u_S(X, X) = |S|X^{|S|-1}$, so both can be evaluated in time $O_{\mathbb{F}}(\log |S|)$. If S is an additive coset in $\mathbb{F}$ then $u_S(X, Y)$ is obtained directly from the linearized polynomial v_S, and $u_S(X, X)$ is the constant polynomial that equals the coefficient of the linear term in the linearized polynomial v_S.

Knowledge soundness. HOL has knowledge error κ if there exists a probabilistic polynomial-time extractor $\mathbf{E}$ such that for every instance $\mathbb{i}$, witness $\mathbb{x}$, and unbounded prover $\tilde{\mathbf{P}}$:

$$\Pr\left[(\mathbb{i},\mathbb{x},\mathbb{w}) \notin \mathcal{R} \;\middle|\; \mathbb{w} \leftarrow \mathbf{E}^{\tilde{\mathbf{P}}}(\mathbb{i},\mathbb{x})\right] \geq \Pr\left[\langle \tilde{\mathbf{P}}, \mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})\rangle = 1\right] - \kappa \ ,$$

where $\mathbf{E}^{\tilde{\mathbf{P}}}$ means that we give the extractor *black-box* access to $\tilde{\mathbf{P}}$. We provide this definition purely for illustrative purposes. In this paper we only use the stronger knowledge notions described in Section 5.4.2.

Zero knowledge. HOL has (perfect) zero knowledge with query bound b if there exists a probabilistic polynomial-time simulator $\mathbf{S}$ such that for every $(\mathbb{i},\mathbb{x},\mathbb{w}) \in \mathcal{R}$ and b-query algorithm $\tilde{\mathbf{V}}$ the random variables $\mathrm{View}(\mathbf{P}(\mathbb{i},\mathbb{x},\mathbb{w}),\tilde{\mathbf{V}})$ and $\mathbf{S}^{\tilde{\mathbf{V}}}(\mathbb{i},\mathbb{x})$, defined below, are identical. (An algorithm is b-query if it makes *less than* b queries in total to any oracles it has access to.)

- $\mathrm{View}(\mathbf{P}(\mathbb{i},\mathbb{x},\mathbb{w}),\tilde{\mathbf{V}})$ is the *view* of $\tilde{\mathbf{V}}$, i.e., is the random variable $(r, a_1, \ldots, a_q)$ where r is $\tilde{\mathbf{V}}$'s randomness and $a_1, \ldots, a_q$ are the responses to $\tilde{\mathbf{V}}$'s queries determined by the oracles sent by $\mathbf{P}$.
- $\mathbf{S}^{\tilde{\mathbf{V}}}(\mathbb{i},\mathbb{x})$ is the output of $\mathbf{S}(\mathbb{i},\mathbb{x})$ when given straightline access to $\tilde{\mathbf{V}}$ ($\mathbf{S}$ may interact with $\tilde{\mathbf{V}}$, *without rewinding*, by exchanging messages with $\tilde{\mathbf{V}}$ and answering its oracle queries), *prepended* with $\tilde{\mathbf{V}}$'s randomness r. Note that r could be of super-polynomial size, so $\mathbf{S}$ cannot sample r on $\tilde{\mathbf{V}}$'s behalf and then output it; instead, we restrict $\mathbf{S}$ to not see r, and prepend r to $\mathbf{S}$'s output.

HOL is *honest-verifier* zero knowledge if the above holds with $\tilde{\mathbf{V}} := \mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$.

Public coins. HOL is *public-coin* if each verifier message to the prover is a random string. This means that the verifier's randomness is its messages $\rho_1, \ldots, \rho_{\mathsf{k}} \in \{0,1\}^*$ and possibly additional randomness $\rho_{\mathsf{k}+1} \in \{0,1\}^*$ used after the interaction. All verifier queries can be postponed, without loss of generality, to a query phase that occurs after the interactive phase with the prover.

5.4.1 Reed–Solomon encoded holographic IOPs

Recall that in an RS-encoded IOP, the prover and verifier engage in a public-coin IOP interaction and, after the interaction, the verifier outputs a set of algebraic claims about the prover's messages. The completeness condition requires that in the "yes" case, when the verifier interacts with the honest prover, the output claims are true with probability 1. The soundness condition requires that in the "no" case, when the verifier interacts with a malicious prover, at least one of the output claims will be false with high probability no matter what the prover's messages are. The holographic setting introduces the sole difference that the verifier's algebraic claims may include statements about the codewords output by the indexer.

In more detail, by "algebraic claim" we specifically mean a *rational constraint*, defined next; this definition is identical to Definition 2.3.5 except that we use degree instead of rate.

Definition 5.4.1. *A* **rational constraint** *is a tuple $\mathcal{C} = (N, D, d)$ where $N\colon \mathbb{F}^{1+\ell} \to \mathbb{F}$ and $D\colon \mathbb{F} \to \mathbb{F}$ are arithmetic circuits, and $d \in \mathbb{N}$ is a degree bound. The arithmetic circuits (N, D)*

and a list of words $f_1, \ldots, f_\ell \colon L \to \mathbb{F}$ *jointly define the word* $(N, D)[f_1, \ldots, f_\ell] \colon L \to \mathbb{F}$ *given by*

$$\forall\, a \in L\,,\ (N, D)[f_1, \ldots, f_\ell](a) := \frac{N(a, f_1(a), \ldots, f_\ell(a))}{D(a)}\ .$$

A rational constraint $\mathcal{C} = (N, D, d)$ *is* **satisfied with respect to** $f_1, \ldots, f_\ell$ *if* $(N, D)[f_1, \ldots, f_\ell] \in \mathrm{RS}\,[L, d]$.[12]

When describing rational constraints, we will often use the shorthand notation "$\deg(\hat{f}) \leq d$", where $f \colon L \to \mathbb{F}$ is defined as a rational equation over some oracles. This should be taken to mean the rational constraint $\mathcal{C} = (N, D, d)$ that is naturally induced by the expression that defines f.

A special type of rational constraint is a *boundary constraint*, defined next.

Definition 5.4.2. *A* **boundary constraint** *is a rational constraint that expresses a condition such as "*$\hat{f}(\alpha) = \beta$*" for some word* $f \colon L \to \mathbb{F}$ *and elements* $\alpha, \beta \in \mathbb{F}$*. Such a condition is represented via the rational constraint* $\mathcal{C} = (N, D, \deg(\hat{f}) - 1)$ *where* $N(X, Y) := Y - \beta$ *and* $D(X) := X - \alpha$*, which can be summarized as "*$\deg(\hat{g}) \leq \deg(\hat{f}) - 1$*" where* $g(a) := (f(a) - \beta)/(a - \alpha)$*. We denote this constraint simply by "*$\hat{f}(\alpha) = \beta$*".*

In the following we use $\mathrm{RS}\,[L, (d_1, \ldots, d_k)] \subseteq (\mathbb{F}^k)^L$ to denote the interleaved Reed–Solomon code over L with degree bounds $(d_1, \ldots, d_k)$, i.e., the set of $k \times |L|$ matrices where the i-th row is a codeword of $\mathrm{RS}\,[L, d_i]$ (which itself is all evaluations over L of univariate polynomials of degree at most d_i).

A *Reed–Solomon encoded holographic IOP* (RS-hIOP) for an indexed relation $\mathcal{R}$ is a tuple

$$(\mathbf{I}, P, V, \{\vec{d}_{\mathbf{I}}, \vec{d}_{\mathbf{P},1}, \ldots, \vec{d}_{\mathbf{P},\mathtt{k}}\})$$

where $\mathbf{I}$ is a deterministic algorithm, P and V are probabilistic interactive algorithms, and $\vec{d}_{\mathbf{I}} \in \mathbb{N}^{\ell_0}, \vec{d}_{\mathbf{P},i} \in \mathbb{N}^{\ell_i}$ are vectors of degree bounds, that satisfies the following properties.

Degree bounds: On input any $\mathtt{i}$, the indexer $\mathbf{I}$ outputs a codeword of $\mathrm{RS}\left[L, \vec{d}_{\mathbf{I}}\right]$. Moreover, on input any $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$ and for every round i, the i-th message of $P(\mathtt{i}, \mathtt{x}, \mathtt{w})$ is a codeword of $\mathrm{RS}\left[L, \vec{d}_{\mathbf{P},i}\right]$.

Completeness: For every $(\mathtt{i}, \mathtt{x}, \mathtt{w}) \in \mathcal{R}$, all rational constraints output by $\mathbf{V}^{\mathbf{I}(\mathtt{i})}(\mathtt{x})$ after interacting with $P(\mathtt{i}, \mathtt{x}, \mathtt{w})$ are satisfied with respect to $\mathbf{I}(\mathtt{i})$ and $P(\mathtt{i}, \mathtt{x}, \mathtt{w})$'s messages with probability 1.

Soundness: For every $(\mathtt{i}, \mathtt{x}) \notin \mathcal{L}(\mathcal{R})$ and unbounded malicious prover $\tilde{\mathbf{P}}$ whose i-th message is a codeword of $\mathrm{RS}\left[L, \vec{d}_{\mathbf{P},i}\right]$, all rational constraints output by $\mathbf{V}^{\mathbf{I}(\mathtt{i})}(\mathtt{x})$ after interacting with $\tilde{\mathbf{P}}$ are satisfied with respect to $\mathbf{I}(\mathtt{i})$ and the prover's messages with probability at most ϵ.

[12] For $a \in L$, if $D(a) = 0$ then we define $(N, D)[f_1, \ldots, f_\ell](a) := \perp$. Note that if this holds for some $a \in L$ then, for *any* words $f_1, \ldots, f_\ell$ and degree bound d, the rational constraint (N, D, d) is not satisfied by $f_1, \ldots, f_\ell$.

Often we will write that V "accepts", which means that all of the rational constraints it outputs are satisfied, or that it "rejects", which means that at least one rational constraint is not satisfied.

We conclude by discussing useful complexity measures for RS-encoded IOPs.

- The **query evaluation time** t_{q} is the natural complexity measure for $\mathbf{V}$, and equals the sum of the query evaluation times of the rational constraints output by $\mathbf{V}$. The query evaluation time of a single rational constraint $\mathcal{C} = (N, D, d)$ is the time required to compute $(N, D)[f_1, \ldots, f_\ell](a) \in \mathbb{F}$ given $a \in L$ and oracle access to $f_1, \ldots, f_\ell$ (and possibly additional information provided by the indexer). That is, it is the time needed to (construct and) evaluate the arithmetic circuits (N, D) at a single point.

- The **maximum degree** is a pair $(d_{\mathsf{c}}, d_{\mathsf{e}}) \in \mathbb{N} \times \mathbb{N}$ defined as follows.

  d_{c} is the "constraint degree", defined as the maximum specified degree of any oracle sent by the prover and any constraint output by the verifier, i.e., $d_{\mathsf{c}} := \max \vec{d}_{\mathbf{P},1} \cup \cdots \cup \vec{d}_{\mathbf{P},\mathsf{k}} \cup \{d : V \text{ outputs } (N, D, d)\}$.

  d_{e} is the "effective degree", which is a quantity arising from the compilation from an RS-encoded IOP to a standard IOP via low-degree testing that is defined as follows:

$$d_{\mathsf{e}} := \max\{d_{\mathsf{c}}\} \cup \left\{\deg(N; \vec{d}_{\mathbf{I}}, \vec{d}_{\mathbf{P},1}, \ldots, \vec{d}_{\mathbf{P},\mathsf{k}}), d + \deg(D) : V \text{ outputs } (N, D, d)\right\}$$

  where $\deg(P; \vec{d})$ for an arithmetic circuit $P\colon \mathbb{F}^{1+m} \to \mathbb{F}$ and degree bounds $d \in \mathbb{F}^m$ denotes the degree of the composed polynomial $P(X, Q_1(X), \ldots, Q_m(X))$ when $\deg(Q_i) = d_i$. Note that $d_{\mathsf{e}} \geq d_{\mathsf{c}}$.

5.4.2 Stronger notions of soundness

Aside from the standard notion of soundness above, there are two further soundness notions that arise when constructing non-interactive arguments from IOPs. These are round-by-round soundness [60, 67] and state-restoration soundness [32], adapted to holographic IOPs. We discuss these below.

Round-by-round soundness. We begin by defining the notion of a (partial) transcript of an IOP, which means all proof strings and verifier messages up to a point where the prover is about to move.

Definition 5.4.3. *A* **transcript** tr *of a holographic IOP* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *is a tuple* $(\Pi_1, m_1, \ldots, \Pi_i, m_i)$ *for some* $i \in [\mathsf{k}]$*, where each* Π_j *is a prover (oracle) message and each* m_j *is a verifier message. We denote the empty transcript by* $\emptyset$*. A transcript is* **full** *if* $i = \mathsf{k}$*, where* k *is the round complexity of* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$.

A protocol $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ has *round-by-round soundness error* ϵ_{rbr} if there exists a function State from the set of transcripts to $\{\mathsf{accept}, \mathsf{reject}\}$ such that for every transcript tr:

- if $(\mathbb{i}, \mathbb{x}) \notin \mathcal{L}(\mathcal{R})$ and $\mathsf{tr} = \emptyset$, then $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$;
- if $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$, then $\mathsf{rbr}(\mathsf{tr}) \leq \epsilon_{\mathrm{rbr}}$ where

$$\mathsf{rbr}(\mathsf{tr}) := \max_{\Pi} \Pr_{m}\left[\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr} \| \Pi \| m) = \mathsf{accept}\right] \quad ;$$

- if $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$ and tr is a full transcript, then $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x}; \mathsf{tr})$ rejects.

The notion of round-by-round soundness for *RS-encoded* holographic IOPs is as above, except that the maximum in the definition of rbr is taken over $\Pi_i \in \mathrm{RS}\left[L, \vec{d}_{\mathbf{P},i}\right]$, for tr a transcript of $i-1$ rounds, and the third condition above need only hold for full transcripts tr where $\Pi_i \in \mathrm{RS}\left[L, \vec{d}_{\mathbf{P},i}\right]$ for all i. In particular, it suffices to define $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr})$ only for tr where the prover messages are of the prescribed degrees; otherwise, the state can be taken to be accept.

Finally we recall the definition of round-by-round knowledge soundness from [67].

Definition 5.4.4. *A holographic IOP* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *for a relation* $\mathcal{R}$ *has* **round-by-round knowledge error** κ_{rbr} *if there exists a function* State *and a polynomial-time extractor* $\mathbf{E}$ *such that for every index* $\mathbb{i}$*, instance* $\mathbb{x}$*, transcript* tr *such that* $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$*, and every oracle message* Π*, if* $\Pr_m[\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr} \| \Pi \| m) = \mathsf{accept}] > \kappa_{\mathrm{rbr}}$ *then* $(\mathbb{i}, \mathbb{x}, \mathbf{E}(\mathbb{i}, \mathbb{x}, \mathsf{tr} \| \Pi)) \in \mathcal{R}$.

For RS-encoded holographic IOPs we relax the above so that $\mathbf{E}$ need only succeed for tr where the prover messages are of the prescribed degrees.

State-restoration soundness. State-restoration soundness captures the ability of the prover to cheat when it is able to rewind the verifier a bounded number of times. State-restoration soundness essentially exactly captures the soundness of non-interactive arguments derived from IOPs via the BCS transformation [32]. In Section 5.10 we prove that this continues to be true when we modify the BCS transform to construct *preprocessing* non-interactive arguments from *holographic* IOPs. However here we do not define state-restoration soundness because in our proof of the compiler we will rely on the BCS transformation as a black box. We note only that if a protocol has round-by-round soundness error ϵ_{rbr} then it has state-restoration soundness error $\epsilon_{\mathrm{sr}}(t) \leq t \cdot \epsilon_{\mathrm{rbr}}$, where t is the bound on the number of rewinds. This fact is relevant because in Section 5.8 we prove that our efficient holographic IOP for R1CS has small round-by-round soundness error.

Knowledge soundness is somewhat delicate and so we discuss it in more detail. In this work we will use the following (fairly strong) definition for state-restoration knowledge soundness.

Definition 5.4.5. *An IOP* $(\mathbf{P}, \mathbf{V})$ *for a relation* $\mathcal{R}$ *has* **state restoration knowledge error** κ_{sr} *if there exists a polynomial-time extractor* $\mathbf{E}$ *such that for all* $\mathbb{i}, \mathbb{x}$ *and every state-restoration prover* $\tilde{\mathbf{P}}$*,*

$$\Pr\left[\begin{array}{c|c} \mathsf{tr}_{\mathrm{sr}} \textit{ is accepting} \wedge & \mathsf{tr}_{\mathrm{sr}} \leftarrow \langle \tilde{\mathbf{P}}, \mathbf{V} \rangle_{\mathrm{sr}} \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} & \mathbb{w} \leftarrow \mathbf{E}(\mathbb{i}, \mathbb{x}, \mathsf{tr}_{\mathrm{sr}}) \end{array}\right] \leq \kappa_{\mathrm{sr}} \ .$$

Note that the power of the extractor is limited compared to the definition given in [32]; in particular, we do not allow the extractor even black-box access to the prover itself, only to a (state-restoration) transcript.

If $(\mathbf{P}, \mathbf{V})$ has round-by-round knowledge error κ_{rbr} then it has state-restoration knowledge error $\kappa_{\mathrm{sr}}(t) \leq t \cdot \kappa_{\mathrm{rbr}}$. This can be seen as follows: recall that a state-restoration transcript $\mathsf{tr}_{\mathrm{sr}}$ consists of a collection of partial (standard) transcripts generated via the state restoration game. The state restoration extractor $\mathbf{E}$ applies the RBR extractor to every partial transcript in $\mathsf{tr}_{\mathrm{sr}}$, and outputs the first valid witness it obtains. Since $\mathsf{tr}_{\mathrm{sr}}$ is accepting and the empty transcript is rejecting, at least one such partial transcript changes state from reject to accept. Round-by-round knowledge error implies that if the RBR extractor fails for this transcript, this happens with probability at most κ_{rbr}; the result follows by a union bound.

5.5 Sumcheck for rational functions

We describe how to extend the univariate sumcheck protocol of Section 3.4 from univariate *polynomials* to univariate *rational functions*. We thus obtain a protocol for checking the value of the sum of a rational function $\hat{N}(X)/\hat{D}(X) \in \mathbb{F}(X)$ over a subgroup K of $\mathbb{F}$.

Definition 5.5.1. *Let $\mathcal{R}$ be the set of all pairs $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, K, d_N, d_D, \sigma), (N, D)\big)$ such that $N \in \mathrm{RS}[L, d_N]$, $D \in \mathrm{RS}[L, d_D]$, and $\hat{D}(a) \neq 0$ for all $a \in K$. The promise relation $\mathcal{R}_{\mathrm{RSUM}} = (\mathcal{R}^{\mathrm{YES}}_{\mathrm{RSUM}}, \mathcal{R}^{\mathrm{NO}}_{\mathrm{RSUM}})$ is defined as follows: $\mathcal{R}^{\mathrm{YES}}_{\mathrm{RSUM}}$ is the subset of pairs in $\mathcal{R}$ such that $\sum_{a \in K} \hat{N}(a)/\hat{D}(a) = \sigma$, and $\mathcal{R}^{\mathrm{NO}}_{\mathrm{RSUM}} := \mathcal{R} \setminus \mathcal{R}^{\mathrm{YES}}_{\mathrm{RSUM}}$.*

Let $\sigma \in \mathbb{F}$ be the claimed value for the sum. We know from Lemma 5.3.4 that a polynomial $\hat{f}(X)$ of degree at most $|K| - 1$ sums to σ over K if and only if there exists a polynomial $\hat{g}(X)$ with degree at most $|K| - 2$ such that $\Sigma_K(\hat{g}, \sigma)(X)$ equals $\hat{f}(X)$. While we do not know how to obtain an equivalence like this one for the case of rational functions, there is a natural approach to build on the case of polynomials.

The prover computes the *polynomial* $\hat{f}(X)$ of minimal degree that agrees with the *rational function* $\hat{N}(X)/\hat{D}(X)$ on K, and then sends (the evaluation of) the corresponding polynomial $\hat{g}$ guaranteed by Lemma 5.3.4 (i.e., such that $\Sigma_K(\hat{g}, \sigma)(X) \equiv \hat{f}(X)$). The verifier can check that $\Sigma_K(\hat{g}, \sigma)(X)$ sums to σ via the rational constraint "$\deg(\hat{g}) \leq |K| - 2$". Then the verifier is left to check that $\Sigma_K(\hat{g}, \sigma)(X)$ agrees with $\hat{N}(X)/\hat{D}(X)$ on K, which is equivalent to checking that $\Sigma_K(\hat{g}, \sigma)(X)\hat{D}(X) - \hat{N}(X)$ vanishes on K (as $\hat{D}(a) \neq 0$ for all $a \in K$), which can be done via a standard use of a second rational constraint.

Construction 5.5.2 (rational sumcheck)**.** *Let $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, K, d_N, d_D, \sigma), (N, D)\big)$ be a pair in $\mathcal{R}$. In the rational sumcheck protocol, the honest prover $\mathbf{P}$ receives as input $(\mathbb{x}, \mathbb{w})$, and sends a codeword $g \in \mathrm{RS}[L, |K| - 2]$ that is obtained as follows: compute the unique polynomial $\hat{f}$ of degree at most $|K| - 1$ that agrees with $\hat{N}(X)/\hat{D}(X)$ on K; compute the unique polynomial $\hat{g}(X)$ of degree at most $|K| - 2$ such that $\Sigma_K(\hat{g}, \sigma)(X) \equiv \hat{f}(X)$; evaluate $\hat{g}(X)$ over L to obtain g. The honest verifier $\mathbf{V}$ receives as input $\mathbb{x}$, and outputs the following two rational constraints: "$\deg(\hat{g}) \leq |K| - 2$" and "$\deg(\hat{e}) \leq d_e$", where $e \colon L \to \mathbb{F}$ is a function and $d_e \in \mathbb{N}$ is a degree bound that are defined as follows:*

$$\forall\, a \in L\,,\ e(a) := \frac{\Sigma_K(g, \sigma)(a) \cdot D(a) - N(a)}{v_K(a)} \quad \textit{and} \quad d_e := \max(d_N, |K| - 1 + d_D) - |K| \ . \tag{5.1}$$

Formally, the above is an *RS-encoded PCP of proximity* for $\mathcal{R}_{\mathrm{RSUM}}$ (see Section 2.3.3.1). For simplicity, in the lemma below we directly establish the properties that we need without this abstraction.

Lemma 5.5.3. *Let $(\mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, K, d_N, d_D, \sigma), (N, D)\big) \in \mathcal{R}$ be such that $L \cap K = \emptyset$, K is a subgroup of $\mathbb{F}$, and $\max(d_N, |K| - 1 + d_D) < |L|$. The protocol $(\mathbf{P}, \mathbf{V})$ in Construction 5.5.2 satisfies the following.*

1. Completeness: *if $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}^{\mathrm{YES}}_{\mathrm{RSUM}}$ then $\mathbf{V}(\mathbb{x})$ outputs rational constraints that are satisfied by (N, D, g), where g is the oracle sent by the honest prover $\mathbf{P}(\mathbb{x}, \mathbb{w})$.*

2. Soundness: *if* $(\mathbb{x}, \mathbb{w}) \in \mathcal{R}^{\text{NO}}_{\text{RSUM}}$ *then for every malicious prover* $\tilde{\mathbf{P}}$ *at least one of the rational constraints output by* $\mathbf{V}(\mathbb{x})$ *is not satisfied by* (N, D, g)*, where* g *is the oracle sent by the malicious prover* $\tilde{\mathbf{P}}$.

The protocol has constraint degree $\max(d_N - |K|, d_D - 1, |K| - 2)$ *and effective degree* $\max(d_N, |K| - 1 + d_D)$*. The query evaluation time of the verifier is* $O_{\mathbb{F}}(\log |K|)$.

Proof. We first argue completeness and then soundness.

Completeness. Suppose that $\sum_{a \in K} \hat{N}(a)/\hat{D}(a) = \sigma$. The honest prover $\mathbf{P}$ sends the polynomial $\hat{g}(X)$ with degree at most $|K| - 2$ such that $\Sigma_K(\hat{g}, \sigma)(X)$ agrees with $\hat{N}(X)/\hat{D}(X)$ on K; the existence of $\hat{g}(X)$ is guaranteed by Lemma 5.3.4. Since $\hat{D}(a) \neq 0$ for all $a \in K$, we also have that $\Sigma_K(\hat{g}, \sigma)(a) \cdot \hat{D}(a) = \hat{N}(a)$ for all $a \in K$. Thus the polynomial $\Sigma_K(\hat{g}, \sigma)(X) \cdot \hat{D}(X) - \hat{N}(X)$ is divisible by the vanishing polynomial $v_K(X)$. We conclude that the two rational constraints "$\deg(\hat{g}) \leq |K| - 2$" and "$\deg(\hat{e}) \leq d_e$" are satisfied.

Soundness. Suppose that $\sum_{a \in K} \hat{N}(a)/\hat{D}(a) \neq \sigma$. Let g be the oracle sent by $\tilde{\mathbf{P}}$ and suppose that the rational constraint "$\deg(\hat{g}) \leq |K| - 2$" is satisfied (or else we are done). By Lemma 5.3.4 we know that $\sum_{a \in K} \Sigma_K(\hat{g}, \sigma)(a) = \sigma$. Hence there must exist $a^* \in K$ such that $\Sigma_K(\hat{g}, \sigma)(a^*) \cdot \hat{D}(a^*) \neq \hat{N}(a^*)$, so a^* is not a root of the polynomial $\Sigma_K(\hat{g}, \sigma)(X) \cdot \hat{D}(X) - \hat{N}(X)$. By definition of e, the polynomials $\Sigma_K(\hat{g}, \sigma)(X) \cdot \hat{D}(X) - \hat{N}(X)$ and $\hat{e}(X) \cdot v_K(X)$ agree on L. Since $d_e + |K| = \max(d_N, |K| - 1 + d_D) < |L|$, if the rational constraint "$\deg(\hat{e}) \leq d_e$" is also satisfied, then we can conclude that these two polynomials are identical, which is a contradiction because a^* is a root of $\hat{e}(X) \cdot v_K(X)$.

Efficiency. The verifier outputs a rational constraint on g, which is the oracle sent by the prover, and a rational constraint on e, which is the virtual oracle defined in Eq. (5.1). So the query evaluation time is dominated by the number of field operations to evaluate e at a single point, which is $O(\log |K|)$ (due to the need to evaluate the vanishing polynomial v_K at that point). The stated constraint and effective degrees can be obtained by keeping track of the degrees of the relevant real and virtual oracles in the protocol (as in the table) and then using the definitions in Section 5.4.1.

oracle	type	constraint degree	numerator degree	denominator degree
g	real	$\lvert K\rvert - 2$	–	–
e	virtual	d_e	$\max(d_N, \lvert K\rvert - 1 + d_D)$	$\lvert K\rvert$

□

Remark 5.5.4 (zero knowledge)**.** In Section 5.6, the above construction will be used as a subprotocol to evaluate the arithmetization of a *public* matrix. For this reason, we do not require any zero knowledge properties of the above construction (and indeed, the construction as described is not zero knowledge). Nonetheless, it is relatively straightforward to obtain a zero knowledge variant of this construction by using bounded independence and zero knowledge sumcheck as in Section 3.4.1.

5.6 Holographic lincheck

We describe a holographic variant of the lincheck protocol (Section 3.5). The *lincheck problem* involves checking linear relations between encodings: given $H \subseteq \mathbb{F}$, Reed–Solomon codewords $f, g \in \mathrm{RS}[L, d]$, and matrix $M \in \mathbb{F}^{H \times H}$, check that $\hat{f}|_H = M \cdot \hat{g}|_H$. Below we consider matrices M that are given in a *sparse representation* (see Definition 5.3.1), as the protocol that we describe leverages this sparsity for efficiency.

Definition 5.6.1. *Let $\mathcal{R}$ be the set of all pairs $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, H, K, d, \langle M \rangle), 1^{\log|K|}, (f_1, f_2)\big)$ such that $\mathbb{F}$ is a finite field, L, H, K are subsets of $\mathbb{F}$, $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ is a sparse representation of a matrix $M \in \mathbb{F}^{H \times H}$, $d \in \mathbb{N}$ is a degree bound, and $f_1, f_2 \in \mathrm{RS}[L, d]$ are codewords. The indexed promise relation $\mathcal{R}_{\mathrm{LIN}} = (\mathcal{R}^{\mathrm{YES}}_{\mathrm{LIN}}, \mathcal{R}^{\mathrm{NO}}_{\mathrm{LIN}})$ is such that $\mathcal{R}^{\mathrm{YES}}_{\mathrm{LIN}}$ is the subset of $\mathcal{R}$ with $\hat{f}_1|_H = M \cdot \hat{f}_2|_H$, and $\mathcal{R}^{\mathrm{NO}}_{\mathrm{LIN}} := \mathcal{R} \setminus \mathcal{R}^{\mathrm{YES}}_{\mathrm{LIN}}$.*

The goal of this section is to prove the following lemma.

Lemma 5.6.2. *Let $(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, L, H, K, d, \langle M \rangle), 1^{\log|K|}, (f_1, f_2)\big) \in \mathcal{R}$ be such that H, K are subgroups of $\mathbb{F}$, $|L| > 3|K| - 3$, and $L \cap (H \cup K) = \emptyset$. The protocol $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ in Construction 5.6.8 satisfies the following.*

1. Completeness: *if $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}^{\mathrm{YES}}_{\mathrm{LIN}}$ then $\mathbf{V}(\mathbb{x})$ outputs rational constraints that are satisfied by (f_1, f_2) and the oracles sent by the honest prover $\mathbf{P}(\mathbb{i}, \mathbb{x})$.*
2. Soundness: *if $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}^{\mathrm{NO}}_{\mathrm{LIN}}$ then for every malicious prover $\tilde{\mathbf{P}}$ all of the rational constraints output by $\mathbf{V}(\mathbb{x})$ are satisfied by (f_1, f_2) and the oracles sent by $\tilde{\mathbf{P}}$ with probability at most $2(|H| - 1)/(|\mathbb{F}| - |H|)$.*

In particular, the construction has two rounds and RBR soundness error $(|H| - 1)/(|\mathbb{F}| - |H|)$. Moreover, the construction can be made zero knowledge; the RBR soundness error is then $|H|/(|\mathbb{F}| - |H|)$.

The prover and indexer run in time $O_{\mathbb{F}}(|L| \log |L|)$, and the verifier's query evaluation time is $O_{\mathbb{F}}(\log |K|)$. The constraint degree is $\max(d - 1, |H| - 2, 2|K| - 3)$ and the effective degree is $\max(|H| - 1 + d, 3|K| - 3)$.

Formally, Construction 5.6.8 is an RS-encoded IOP of proximity for $\mathcal{R}_{\mathrm{LIN}}$. However, for simplicity, we directly establish the properties we need without this abstraction. The notion of zero knowledge is as usual for proximity notions: we require that if a malicious verifier $\tilde{\mathbf{V}}$ makes t queries across all of the oracles available to it, the simulator can reproduce its view by making t queries to *each of* the witness oracles.

The remainder of this section proceeds as follows. In Section 5.6.1, we describe a subprotocol for checking an evaluation of the low-degree (bivariate) extension of a matrix. In Section 5.6.2, we describe how to use this subprotocol to build a holographic lincheck protocol, proving Lemma 5.6.2. Throughout this section we rely on the notion of a sparse representation of a matrix (see Section 5.3.1) and on facts about vanishing polynomials and their derivatives (see Section 5.3.3).

5.6.1 Holographic proof for sparse matrix arithmetization

The *bivariate low-degree extension* of a given matrix $M \in \mathbb{F}^{H \times H}$ is the unique polynomial $\hat{M} \in \mathbb{F}[X, Y]$ of minimal degree such that $\hat{M}(a, b) = M_{a,b}$ for all $a, b \in H$. We wish to check

statements of the form "$\hat{M}(\alpha, \beta) = \gamma$" for α, β chosen (almost) arbitrarily in $\mathbb{F}$.

Definition 5.6.3. *The indexed relation $\mathcal{R}_{\mathrm{MAT}}$ is the set of triples*

$$(\mathbb{i}, \mathbb{x}, \mathbb{w}) = \big((\mathbb{F}, H, K, \langle M \rangle), (\alpha, \beta, \gamma), \bot\big)$$

where $\mathbb{F}$ is a finite field, H and K are subsets of $\mathbb{F}$, $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ is a sparse representation of a matrix $M \in \mathbb{F}^{H \times H}$, $\alpha, \beta, \gamma \in \mathbb{F}$ are field elements, and $\hat{M}(\alpha, \beta) = \gamma$. (This relation has no witnesses.)

The indexed relation $\mathcal{R}_{\mathrm{MAT}}$ is tractable: one can check if $\hat{M}(\alpha, \beta) = \gamma$ in time $O_{\mathbb{F}}(\|M\|)$ by directly computing the value of the low-degree extension $\hat{M}(X, Y)$ at (α, β). Without holography, it is not possible to verify this equation in time $o_{\mathbb{F}}(\|M\|)$ since in general $\hat{M}(\alpha, \beta)$ depends on every entry of M.

We show how to significantly reduce this cost via a protocol that holographically stores information about M in the encoded index in order to achieve an online verification time of $O(\log \|M\|)$. Our protocol relies on expressing the *bivariate* low-degree extension $\hat{M}(X, Y)$ in terms of *univariate* low-degree extensions that describe the non-zero entries of M. We explain this algebraic identity, and then how our protocol uses it.

Given a sparse representation $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$, define $\hat{\mathsf{row}}_{\langle M \rangle}, \hat{\mathsf{col}}_{\langle M \rangle} \colon K \to H, \hat{\mathsf{val}}_{\langle M \rangle} \in \mathbb{F}[X]$ to be the unique polynomials of minimal degree such that for each $k \in K$, letting $(a, b, \alpha) := \langle M \rangle(k)$,

$$\hat{\mathsf{row}}_{\langle M \rangle}(k) := a \ , \quad \hat{\mathsf{col}}_{\langle M \rangle}(k) := b \ , \quad \hat{\mathsf{val}}_{\langle M \rangle}(k) := \frac{\alpha}{u_H(a, a) \cdot u_H(b, b)} \ .$$

The following claim expresses $\hat{M}$ in terms of $\hat{\mathsf{row}}_{\langle M \rangle}, \hat{\mathsf{col}}_{\langle M \rangle}, \hat{\mathsf{val}}_{\langle M \rangle}$.

Claim 5.6.4. *For any sparse representation $\langle M \rangle \colon K \to H \times H \times \mathbb{F}$ of a matrix $M \in \mathbb{F}^{H \times H}$,*

$$\hat{M}(X, Y) \equiv \sum_{k \in K} \frac{v_H(X)}{(X - \hat{\mathsf{row}}_{\langle M \rangle}(k))} \cdot \frac{v_H(Y)}{(Y - \hat{\mathsf{col}}_{\langle M \rangle}(k))} \cdot \hat{\mathsf{val}}_{\langle M \rangle}(k) \ .$$

Proof. Denote the right-hand side of the equation by $P(X, Y)$. Since $\hat{\mathsf{row}}_{\langle M \rangle}(k), \hat{\mathsf{col}}_{\langle M \rangle}(k) \in H$ for all $k \in K$, $P(X, Y)$ is a polynomial of degree at most $|H| - 1$ in both X and Y. We now argue that $P(a, b) = M_{a,b}$ for arbitrary $a, b \in H$ (which implies that P agrees with $\hat{M}$ on $H \times H$ and hence that $P \equiv \hat{M}$). Suppose first that there is no $k \in K, \gamma \in \mathbb{F}$ such that $\langle M \rangle(k) = (a, b, \gamma)$. By definition of M, $M_{a,b} = 0$; moreover for any $k \in K$ either $v_H(X)/(X - \hat{\mathsf{row}}_{\langle M \rangle}(k))$ has a root at a or $v_H(Y)/(Y - \hat{\mathsf{col}}_{\langle M \rangle}(k))$ has a root at b, and so $P(a, b) = 0$ as well. Now suppose that there exists $k \in K, \gamma \in \mathbb{F}$ such that $\langle M \rangle(k) = (a, b, \gamma)$; note that k is unique because $\langle M \rangle$ is injective. Hence $P(a, b) = u_H(a, a) \cdot u_H(b, b) \cdot \hat{\mathsf{val}}_{\langle M \rangle}(k) = M_{a,b}$. □

Construction 5.6.5. *The indexer $\mathbf{I}$ receives as input an index $\mathbb{i} = (\mathbb{F}, H, K, \langle M \rangle)$ along with an evaluation domain $L \subseteq \mathbb{F}$, computes the low-degree extensions $\hat{\mathsf{row}}_{\langle M \rangle}, \hat{\mathsf{col}}_{\langle M \rangle}, \hat{\mathsf{val}}_{\langle M \rangle}$, and then outputs their evaluations $\mathsf{row}_{\langle M \rangle}, \mathsf{col}_{\langle M \rangle}, \mathsf{val}_{\langle M \rangle} \in \mathrm{RS}\,[L, |K| - 1]$. The indexer $\mathbf{I}$ also outputs descriptions of $\mathbb{F}, H, K$.*

Subsequently, given an instance $\mathbb{x} = (\alpha, \beta, \gamma)$*, the honest prover* $\mathbf{P}$ *receives as input* $(\mathbb{i}, \mathbb{x})$ *and the honest verifier* $\mathbf{V}$ *receives as input* $\mathbb{x}$ *and oracle access to* $\mathbf{I}(\mathbb{i})$*. The prover* $\mathbf{P}$ *and verifier* $\mathbf{V}$ *engage in the rational sumcheck protocol (see Section 5.5) to show that*

$$\sum_{k \in K} \frac{v_H(\alpha)}{(\alpha - \hat{\mathsf{row}}_{\langle M \rangle}(k))} \cdot \frac{v_H(\beta)}{(\beta - \hat{\mathsf{col}}_{\langle M \rangle}(k))} \cdot \hat{\mathsf{val}}_{\langle M \rangle}(k) = \gamma \ .$$

In particular, the verifier $\mathbf{V}$ *outputs the rational constraints "*$\deg(\hat{g}) \leq |K| - 2$*" for* g *sent by* $\mathbf{P}$*, and "*$\deg(\hat{e}) \leq 2|K| - 3$*" for* $e \colon L \to \mathbb{F}$ *defined as*

$$\forall\, a \in L\,,\ e(a) := \frac{\Sigma_K(g, \gamma)(a) \cdot (\alpha - \mathsf{row}_{\langle M \rangle}(a))(\beta - \mathsf{col}_{\langle M \rangle}(a)) - v_H(\alpha) v_H(\beta) \mathsf{val}_{\langle M \rangle}(a)}{v_K(a)} \ . \tag{5.2}$$

Lemma 5.6.6. *For any field* $\mathbb{F}$ *and evaluation domain* $L \subseteq \mathbb{F}$*, Construction 5.6.5 is an RS-encoded holographic PCP over domain* L *for the indexed relation* $\mathcal{R}_{\mathrm{MAT}}$ *with perfect completeness and perfect soundness, for indices* $\mathbb{i} = (\mathbb{F}, H, K, \langle M \rangle)$ *and instances* $\mathbb{x} = (\alpha, \beta, \gamma)$ *with* H, K *subgroups of* $\mathbb{F}$*,* $|L| > 3|K| - 3$*,* $L \cap K = \emptyset$*, and* $\alpha, \beta \in \mathbb{F} \setminus H$*. In particular, the following properties hold.*

1. Completeness: *if* $(\mathbb{i}, \mathbb{x}, \bot) \in \mathcal{R}_{\mathrm{MAT}}$ *then* $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ *outputs rational constraints that are satisfied by the oracles sent by the honest prover* $\mathbf{P}(\mathbb{i}, \mathbb{x})$*.*
2. Soundness: *if* $(\mathbb{i}, \mathbb{x}, \bot) \notin \mathcal{R}_{\mathrm{MAT}}$ *then for every malicious prover* $\tilde{\mathbf{P}}$ *at least one of the rational constraints output by* $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x})$ *is not satisfied by the oracles sent by* $\tilde{\mathbf{P}}$*.*

The constraint degree is $2|K| - 3$*, and the effective degree is* $3|K| - 3$*. The indexer and prover run in time* $O_{\mathbb{F}}(|L| \log |L|)$*, and the query evaluation time of the verifier is* $O_{\mathbb{F}}(\log |K|)$*.*

Proof. Completeness and soundness follow immediately from Claim 5.6.4, the completeness and soundness of the rational sumcheck protocol, and the observation that the denominator of the rational summand for $\hat{M}(\alpha, \beta)$ is nonzero for all $k \in K$ when $\alpha, \beta \in \mathbb{F} \setminus H$. The query evaluation time is dominated by the cost of evaluating v_H and v_K at a point, which is $O_{\mathbb{F}}(\log |H| + \log |K|) = O_{\mathbb{F}}(\log |K|)$. The constraint degree and effective degree are obtained from setting $d_N := |K| - 1$ and $d_D := 2|K| - 2$ in the rational sumcheck protocol (see Lemma 5.5.3). □

5.6.2 The protocol

Recall from Section 5.3.3 that $\vec{r} := \big(u_H(a, Y)\big)_{a \in H} \in \mathbb{F}[Y]^H$ is a vector of linearly independent polynomials in Y. The primary computational task in the lincheck protocol is to evaluate the low-degree extension $u_M(X, Y)$ of $\vec{r}M \in \mathbb{F}[Y]^H$ at a uniformly chosen point in $\mathbb{F} \times \mathbb{F}$. For this, we use the protocol for sparse matrix arithmetization discussed above, along with an observation showing that it suffices to compute the arithmetization of a matrix M^* related to M.

Claim 5.6.7. *For any matrix* $M \in \mathbb{F}^{H \times H}$*, let* $M^* \in \mathbb{F}^{H \times H}$ *be the matrix given by* $M^*_{a,b} := M_{b,a} \cdot u_H(b, b)$ *for all* $a, b \in H$*; note that* $\|M^*\| = \|M\|$*. Then*

$$u_M(X, Y) \equiv \hat{M}^*(X, Y) \ .$$

Proof. By the definition of low-degree extension,

$$u_M(X,Y) \equiv \sum_{a\in H} (\vec{r}M)_a \cdot L_{a,H}(X) \equiv \sum_{a\in H} L_{a,H}(X) \sum_{b\in H} M_{b,a} \cdot u_H(b,Y) \ .$$

Recall that $u_H(b,Y) \equiv u_H(b,b)L_{b,H}(Y)$. Hence

$$u_M(X,Y) \equiv \sum_{a\in H}\sum_{b\in H} L_{a,H}(X)L_{b,H}(Y)M_{b,a}u_H(b,b) \equiv \hat{M}^*(X,Y) \ . \qquad \square$$

Construction 5.6.8 (holographic lincheck)**.** *The indexer* $\mathbf{I}$ *receives as input an index* $\mathbb{i} = (\mathbb{F}, L, H, K, d, \langle M\rangle)$*, computes a sparse representation* $\langle M^*\rangle$ *of the matrix* M^* *(as in Claim 5.6.7), and then runs the indexer of the sparse matrix arithmetization protocol (Construction 5.6.5) on the index* $(\mathbb{F}, H, K, \langle M^*\rangle)$*; note that the output of the latter includes descriptions of* $\mathbb{F}, H, K$*.*

Subsequently, given an instance $\mathbb{x} = 1^{\log|K|}$ *and witness* $\mathbb{w} = (f_1, f_2)$*, the honest prover* $\mathbf{P}$ *receives as input* $(\mathbb{i}, \mathbb{x})$*, the honest verifier* $\mathbf{V}$ *receives as input* $\mathbb{x}$ *and oracle access to* $\mathbf{I}(\mathbb{i})$*, and they engage in the following protocol.*

1. $\mathbf{V}$ *sends* $\alpha \in \mathbb{F} \setminus H$ *uniformly at random.*
2. $\mathbf{P}$ *sends the evaluation* $t \in \mathrm{RS}[L, |H|-1]$ *of the polynomial* $\hat{t}(X) := u_M(X,\alpha)$*.*
3. $\mathbf{P}, \mathbf{V}$ *engage in the sumcheck protocol to show that* $\sum_{b\in H} u_H(b,\alpha)\hat{f}_1(b) - \hat{t}(b)\hat{f}_2(b) = 0$*.*
   That is, $\mathbf{P}$ *sends* $g_1 \in \mathrm{RS}[L, |H|-2]$ *and* $\mathbf{V}$ *outputs the rational constraints "*$\deg(\hat{g}_1) \leq |H|-2$*" and "*$\deg(\hat{h}) \leq d-1$*" where*

$$\forall\, b \in L\,,\ h(b) := \frac{u_H(b,\alpha)f_1(b) - t(b)f_2(b) - \Sigma_H(g_1,0)(b)}{v_H(b)} \ .$$

4. $\mathbf{V}$ *sends* $\beta \in \mathbb{F} \setminus H$ *uniformly at random.*
5. $\mathbf{P}$ *sends the field element* $\gamma := u_M(\beta,\alpha) = \hat{t}(\beta)$*, and* $\mathbf{V}$ *outputs the boundary constraint "*$\hat{t}(\beta) = \gamma$*".*
6. $\mathbf{P}, \mathbf{V}$ *engage in the matrix arithmetization protocol (Construction 5.6.5) to show that* $\hat{M}^*(\beta,\alpha) = \gamma$*.*
   That is, $\mathbf{P}$ *sends* $g_2 \in \mathrm{RS}[L, |K|-2]$ *and* $\mathbf{V}$ *outputs the rational constraints "*$\deg(\hat{g}_2) \leq |K|-2$*" and "*$\deg(\hat{e}) \leq 2|K|-3$*", where* $e\colon L \to \mathbb{F}$ *is as in Eq.* (5.2) *with* g_2 *in place of* g *and* M^* *in place of* M*.*

Proof of Lemma 5.6.2. For $\alpha \in \mathbb{F}$, let $\vec{r}_\alpha := \big(u_H(b,\alpha)\big)_{b\in H} \in \mathbb{F}^H$. One can verify that $\vec{r}_\alpha M = \big(u_M(b,\alpha)\big)_{b\in H}$.

Completeness. Suppose that $\hat{f}_1|_H = M \cdot \hat{g}|_H$. Then for every $\alpha \in \mathbb{F} \setminus H$ it holds that

$$\sum_{b\in H} u_H(b,\alpha)\hat{f}_1(b) = \langle \vec{r}_\alpha, \hat{f}_1|_H\rangle = \langle \vec{r}_\alpha M, \hat{f}_2|_H\rangle = \sum_{b\in H} u_M(b,\alpha)\hat{f}_2(b) = \sum_{b\in H} \hat{t}(b)\hat{f}_2(b)$$

and so the sumcheck protocol in Step 3 succeeds. Next, Claim 5.6.7 tells us that $u_M(X,Y) \equiv \hat{M}^*(X,Y)$ and so for every $\beta \in \mathbb{F} \setminus H$ it holds that $u_M(\beta,\alpha) = \hat{M}^*(\beta,\alpha)$. This means that $\hat{M}^*(\beta,\alpha) = \hat{t}(\beta) = \gamma$, and so the matrix arithmetization subverifier accepts, and the boundary constraint "$\hat{t}(\beta) = \gamma$" is satisfied.

Round-by-round soundness. We define the State function as follows.

$\mathsf{State}(M, f_1, f_2, (\alpha, t, g_1, \beta, (\gamma, g_2))):$

1. If $\alpha \neq \perp$ is such that $\langle \vec{r}_\alpha, \hat{f}_1|_H \rangle = \langle \vec{r}_\alpha, M \cdot \hat{f}_2|_H \rangle$, output accept.
2. If $\hat{t}(X) \not\equiv u_M(X, \alpha)$ and $\beta \neq \perp$ is such that $\hat{t}(\beta) = u_M(\beta, \alpha)$, output accept.
3. Otherwise, output reject.

Clearly $\mathsf{State}(\emptyset) = \mathsf{reject}$. Suppose that $\hat{f}_1|_H \neq M \cdot \hat{f}_2|_H$. Then $\mathrm{Pr}_{\alpha \in \mathbb{F} \setminus H}[\langle \vec{r}_\alpha, \hat{f}_1|_H \rangle = \langle \vec{r}_\alpha, M \cdot \hat{f}_2|_H \rangle] \leq (|H| - 1)/(|\mathbb{F}| - |H|)$, and so the probability of moving to accept in the first round is at most $(|H| - 1)/(|\mathbb{F}| - |H|)$. Similarly, if $\hat{t}(X) \not\equiv u_M(X, \alpha)$, then $\mathrm{Pr}_{\beta \in \mathbb{F} \setminus H}[\hat{t}(\beta) = u_M(\beta, \alpha)] \leq (|H| - 1)/(|\mathbb{F}| - |H|)$, and so the probability of moving to accept in the second round is at most $(|H| - 1)/(|\mathbb{F}| - |H|)$.

Now suppose that $\mathsf{State}(M, f_1, f_2, (\alpha, t, g_1, \beta, (\gamma, g_2))) = \mathsf{reject}$; we show that with probability 1 there is some constraint which is not satisfied. By definition of State, $\langle \vec{r}_\alpha, \hat{f}_1|_H \rangle \neq \langle \vec{r}_\alpha, M \cdot \hat{f}_2|_H \rangle$. If $\hat{t}(X) \equiv u_M(X, \alpha)$ then the sumcheck constraint is not satisfied with probability 1, by the soundness of the sumcheck protocol. Otherwise $\hat{t}(X) \not\equiv u_M(X, \alpha)$; then by definition of State, it holds that $\hat{t}(\beta) \neq u_M(\beta, \alpha)$.

Hence with probability 1, either the sumcheck constraint is not satisfied, the boundary constraint "$\hat{t}(\beta) = \gamma$" is not satisfied, or $\gamma = \hat{t}(\beta) \neq u_M(\beta, \alpha) = \hat{M}^*(\beta, \alpha)$, and so the matrix arithmetization subverifier outputs a rational constraint that is not satisfied.

Zero knowledge. Note that since M is part of the index rather than the witness, it is not relevant for zero knowledge; in particular, the simulator has access to M. Hence to ensure zero knowledge we need only modify Step 3 to use the zero knowledge sumcheck protocol. This adds an additional oracle (the random mask) but not an additional round, since we can run the protocols in parallel. The soundness error increases by at most $1/|\mathbb{F}|$ (and since the ZK sumcheck protocol is one round, so does the RBR soundness error). The sumcheck simulator satisfies the property that the view of a malicious verifier making t queries across all oracles can be simulated by making t queries to the summand. Since the summand is defined pointwise with respect to (f_1, f_2), this results in at most t queries to each of f_1, f_2.

Efficiency. The indexer runs the indexer of the holographic protocol for sparse matrix arithmetization on (a modification of) the given matrix, and so runs in time $O_{\mathbb{F}}(|L| \log |L|)$. The first message of the prover is for the sumcheck protocol and the second message of the prover is for the sparse matrix arithmetization protocol, and so the prover runs in time $O_{\mathbb{F}}(|L| \log |L|)$. The query evaluation time of the verifier is dominated by the cost to evaluate the vanishing polynomials v_H and v_K at a point, and so is $O_{\mathbb{F}}(\log |H| + \log |K|) = O_{\mathbb{F}}(\log |K|)$.

The constraint degree is $\max(d-1, |H|-2, 2|K|-3)$ and the effective degree is $\max(|H|-1+d, 3|K|-3)$, as can be seen by keeping track of the degrees of all relevant real and virtual oracles in the protocol (as in the table) and then using the definitions in Section 5.4.1.

oracle	type	constraint degree	numerator degree	denominator degree
g_1	real	$\|H\| - 2$	–	–
h	virtual	$d - 1$	$\|H\| - 1 + d$	$\|H\|$
g_2	real	$\|K\| - 2$	–	–
e	virtual	$2\|K\| - 3$	$3\|K\| - 3$	$\|K\|$

□

5.7 RS-encoded holographic IOP for R1CS

We describe an RS-encoded holographic IOP for R1CS. The main subroutine that we use is the RS-encoded holographic protocol for lincheck that we obtained in Section 5.6.

Theorem 5.7.1. *Fix some $L \subseteq \mathbb{F}$ and $\mathsf{b} \in \mathbb{N}$. Construction 5.7.2 below is a RS-encoded holographic IOP of knowledge for $\mathcal{R}_{\mathrm{R1CS}}$ (Definition 5.3.2) over domain L for indices $(\mathbb{F}, H, K, \langle A\rangle, \langle B\rangle, \langle C\rangle)$ such that H, K are subgroups of $\mathbb{F}$, $|L| > 3|K| - 3$, and $L \cap (H \cup K) = \emptyset$. The protocol has 5 messages (prover moves first), is zero knowledge against verifiers making less than b queries, and has round-by-round knowledge error $|H|/(|\mathbb{F}| - |H|)$. The index length is $O_{\mathbb{F}}(|L|)$, and the proof length is $O_{\mathbb{F}}(|L|)$. The prover and indexer run in time $O_{\mathbb{F}}(|L| \log |L|)$ and the verifier runs in time $O_{\mathbb{F}}(|v| + \log |K|)$. The constraint degree is $\max(|H| + \mathsf{b} - 2, 2|K| - 3)$, and the effective degree is $\max(2|H| + \mathsf{b} - 2, 3|K| - 3)$.*

Construction 5.7.2. *We describe an RS-encoded holographic IOP $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ for R1CS. (See Fig. 5.4 for a diagram of this protocol after applying optimizations described in Remark 5.7.3 below.) In the description below we denote by $(\mathbf{I}_{\mathrm{LIN}}, \mathbf{P}_{\mathrm{LIN}}, \mathbf{V}_{\mathrm{LIN}})$ the zero knowledge holographic protocol for lincheck (Construction 5.6.8).*

The indexer $\mathbf{I}$ receives as input an index $\mathbb{i} = (\mathbb{F}, H, K, \langle A\rangle, \langle B\rangle, \langle C\rangle)$, computes the encoded index $\mathbb{I}_M \leftarrow \mathbf{I}_{\mathrm{LIN}}(\mathbb{i}^M_{\mathrm{LIN}})$ where $\mathbb{i}^M_{\mathrm{LIN}} := (\mathbb{F}, L, H, K, |H| + \mathsf{b} - 1, \langle M\rangle)$ for each $M \in \{A, B, C\}$ for each $M \in \{A, B, C\}$, and then outputs the tuple $(\mathbb{I}_A, \mathbb{I}_B, \mathbb{I}_C)$. (Implicitly this includes descriptions of $\mathbb{F}, H, K$; recall also that in an RS-encoded protocol all parties have access to a description of the evaluation domain L.)

Subsequently, the prover $\mathbf{P}$ receives as input the index $\mathbb{i}$, an instance $\mathbb{x} = (I, v)$, and a witness $\mathbb{w} = w$; the verifier $\mathbf{V}$ receives as input the instance $\mathbb{x}$ only. Let $z := (v, w) \in \mathbb{F}^H$ be the full variable assignment.

1. ***Compute LDE of the input.*** *Before the interaction, the prover $\mathbf{P}$ constructs $\hat{f}_v(X)$, the unique polynomial of degree less than $|I|$ such that, for all $b \in I$, $\hat{f}_v(b) = v_b$. Define $f_v := \hat{f}_v|_L$. Note that the verifier $\mathbf{V}$, which knows v, can evaluate $\hat{f}_v(X)$ at any point in $\mathbb{F}$ in time $O_{\mathbb{F}}(|v|)$.*
2. ***Witness and auxiliary oracles.*** *The prover $\mathbf{P}$ sends to the verifier $\mathbf{V}$ the oracles*

$$f_w \in \mathrm{RS}\,[L, |H| - |I| + \mathsf{b} - 1] \quad \textit{and} \quad f_{Az}, f_{Bz}, f_{Cz} \in \mathrm{RS}\,[L, |H| + \mathsf{b} - 1]$$

defined as follows.

- *$f_w := \bar{f}_w|_L$ where $\bar{f}_w$ is a random polynomial of degree less than $|H| - |I| + \mathsf{b}$ such that*

$$\forall\, a \in H \setminus I, \quad \bar{f}_w(a) = \frac{w_a - \hat{f}_v(a)}{v_I(a)} \ .$$

- *$f_{Az} := \bar{f}_{Az}|_L$ where $\bar{f}_{Az}$ is a random polynomial of degree less than $|H| + \mathsf{b}$ such that, for all $a \in H$, $\bar{f}_{Az}(a) = \sum_{b \in H} A_{a,b} \cdot z_b = (Az)_a$. The other codewords, f_{Bz} and f_{Cz}, are defined similarly.*

The codewords f_v and f_w implicitly define the "virtual oracle" $f_z \in \mathrm{RS}\,[L, |H| + \mathsf{b} - 1]$ where $f_z(a) := f_w(a)v_I(a) + f_v(a)$ for $a \in L$. Note that $\hat{f}_z(a) = z_a$ for all $a \in H$, so $\hat{f}_z$ is a low-degree extension of z.

3. ***Rowcheck.*** *To test that* $\hat{f}_{Az}|_H \circ \hat{f}_{Bz}|_H = \hat{f}_{Cz}|_H$*, the verifier* $\mathbf{V}$ *outputs the rational constraint "*$\deg(\hat{s}) \leq |H| + 2\mathsf{b} - 2$*" where* $s\colon L \to \mathbb{F}$ *is defined as*

$$\forall\, a \in L\,,\ s(a) := \frac{f_{Az}(a) \cdot f_{Bz}(a) - f_{Cz}(a)}{v_H(a)}\ .$$

4. ***Linchecks.*** *To test that* $\hat{f}_{Mz}|_H = M \cdot \hat{f}_z|_H$ *for each* $M \in \{A, B, C\}$*, the prover* $\mathbf{P}$ *and verifier* $\mathbf{V}$ *run the following in parallel. Recall that the verifier* $\mathbf{V}$ *has oracle access to the encoded index is* $(\mathbb{I}_A, \mathbb{I}_B, \mathbb{I}_C)$.
   (a) $\left(\mathbf{P}_{\text{LIN}}(\mathbb{i}^A_{\text{LIN}}, 1^{\log|K|}, (f_{Az}, f_z)), \mathbf{V}^{f_{Az}, f_z, \mathbb{I}_A}_{\text{LIN}}(1^{\log|K|})\right)$.
   (b) $\left(\mathbf{P}_{\text{LIN}}(\mathbb{i}^B_{\text{LIN}}, 1^{\log|K|}, (f_{Bz}, f_z)), \mathbf{V}^{f_{Bz}, f_z, \mathbb{I}_B}_{\text{LIN}}(1^{\log|K|})\right)$.
   (c) $\left(\mathbf{P}_{\text{LIN}}(\mathbb{i}^C_{\text{LIN}}, 1^{\log|K|}, (f_{Cz}, f_z)), \mathbf{V}^{f_{Cz}, f_z, \mathbb{I}_C}_{\text{LIN}}(1^{\log|K|})\right)$.

Proof. Fix an index $\mathbb{i} = (\mathbb{F}, H, K, \langle A\rangle, \langle B\rangle, \langle C\rangle)$ and instance $\mathbb{x} = (I, v)$.

Completeness. Suppose that $(\mathbb{i}, \mathbb{x}, w) \in \mathcal{R}_{\text{R1CS}}$, and let $z := (v, w)$. Note that, by construction, $\hat{f}_z$ is a low-degree extension of z. Since $Az \circ Bz = Cz$, we know that $\hat{f}_{Az}|_H \circ \hat{f}_{Bz}|_H = \hat{f}_{Cz}|_H$ and $\hat{f}_{Mz}|_H = M \cdot \hat{f}_z|_H$ for each $M \in \{A, B, C\}$. Hence, for all $a \in H$ it holds that $f_{Az}(a) \cdot f_{Bz}(a) - f_{Cz}(a) = 0$, and so $\hat{s}(X)$ is a polynomial of degree at most $2(|H| + \mathsf{b} - 1) - |H| = |H| + 2\mathsf{b} - 2$, so the rational constraint in Step 3 is satisfied. Moreover, the holographic lincheck protocol in Step 4a yields rational constraints which are satisfied; by a similar argument, Steps 4b and 4c yield satisfied rational constraints.

Round-by-round soundness. We define the following State function ($\mathbb{i} = (\mathbb{F}, H, K, \langle A\rangle, \langle B\rangle, \langle C\rangle)$):

$\mathsf{State}(\mathbb{i}, (I, v), (f_w, f_{Az}, f_{Bz}, f_{Cz}, \mathsf{tr}_{\text{LIN}}))$:
1. If $(\mathbb{i}, (I, v), \hat{f}_w|_{H\setminus I}) \in \mathcal{R}_{\text{R1CS}}$, output accept.
2. Split tr_{LIN} into (partial) transcripts $\mathsf{tr}^A_{\text{LIN}}, \mathsf{tr}^B_{\text{LIN}}, \mathsf{tr}^C_{\text{LIN}}$ for the three lincheck subprotocols on A, B, C respectively. If there exists $M \in \{A, B, C\}$ such that $\hat{f}_{Mz}|_H \neq M \cdot \hat{f}_z|_H$ but the state function for the lincheck protocol accepts $(M, f_{Mz}, f_z, \mathsf{tr}_{\text{LIN}})$, output accept.
3. Otherwise, output reject.

Suppose that $(\mathbb{i}, (I, v)) \notin \mathcal{L}(\mathcal{R}_{\text{R1CS}})$. Item 1 never holds. By the round-by-round soundness of the lincheck protocol, the probability that State moves to accept at any round is at most $|H|/(|\mathbb{F}| - |H|)$. It remains to show that when $\mathsf{State}(\mathbb{i}, (I, v), (f_w, f_{Az}, f_{Bz}, f_{Cz}, \mathsf{tr}_{\text{LIN}})) = \mathsf{reject}$, the verifier rejects with probability 1.

Let f_z be the virtual oracle induced by f_w as sent by $\tilde{\mathbf{P}}$. Then either $\hat{f}_{Az}|_H \circ \hat{f}_{Bz}|_H \neq \hat{f}_{Cz}|_H$, or there exists $M \in \{A, B, C\}$ such that $\hat{f}_{Mz}|_H \neq M \cdot \hat{f}_z|_H$. In the former case, the rational constraint output in Step 3 is not satisfied (with probability 1), so suppose that the latter holds. Then by the definition of State, the state function for the lincheck protocol rejects for some such M. Hence some rational constraint output by the verifier in Steps 4a, 4b and 4c is not satisfied.

Proof of knowledge. The extractor $\mathbf{E}(\mathbb{i}, \mathbb{x}, \mathsf{tr})$ operates as follows: read f_w from tr and output $\hat{f}_w|_{H\setminus I}$. Let $S := \{f_w : (\mathbb{i}, \mathbb{x}, \hat{f}_w|_{H\setminus I}) \in \mathcal{R}_{\text{R1CS}}\}$. From the round-by-round soundness analysis, it holds that conditioned on $f_w \notin S$, the state moves to accept with probability at most $\epsilon := |H|/(|\mathbb{F}| - |H|)$. Hence if the state moves to accept with probability greater than ϵ, it must be that $f_w \in S$, and so $\mathbf{E}$ succeeds.

Zero knowledge. The simulator $\mathbf{S}$ simulates the oracles $f_w, f_{Az}, f_{Bz}, f_{Cz}$ by answering $\tilde{\mathbf{V}}$'s queries with uniformly random elements of $\mathbb{F}$. It runs the simulator for the zero knowledge holographic lincheck protocol as appropriate, answering the subsimulators' queries to the oracles with uniformly random field elements. Since $\tilde{\mathbf{V}}$ makes $t < \mathsf{b}$ queries across all oracles, the guarantees of the subsimulators ensure that we only need to simulate at most t evaluations of each of $f_w, f_{Az}, f_{Bz}, f_{Cz}$ in L (with $L \cap H = \emptyset$), which by bounded independence properties of random polynomials will be uniformly random elements of $\mathbb{F}$. For a detailed simulator construction for a similar protocol, see Section 3.7.1.

Efficiency. The running time of the indexer follows from the running time of the lincheck indexer; in particular, its computation cost is dominated by the cost of a constant number of FFTs over L. The running time of the prover is similarly dominated. The constraint cost of the verifier consists of evaluating the low degree extension of v at a single point in $\mathbb{F}$, and running the lincheck and rowcheck subverifiers whose cost is dominated by evaluating v_H and v_K; using preprocessing this can be achieved for H, K subgroups of $\mathbb{F}$ in time $O_{\mathbb{F}}(\log |K|)$. □

Remark 5.7.3 (batching linchecks)**.** We would like to batch the three lincheck protocols from Steps 4a to 4c into a single one, similarly to what is done in the non-holographic protocol for R1CS. Informally, we want the verifier to send random elements η_A, η_B, η_C and then run a holographic lincheck for the matrix $\eta_A A + \eta_B B + \eta_C C$. However doing this requires some care because the verifier only has access to the encoded indices $(\mathbb{I}_A, \mathbb{I}_B, \mathbb{I}_C)$ for the matrices A, B, C, as opposed to an encoded index for the matrix $\eta_A A + \eta_B B + \eta_C C$, and our holographic lincheck protocol is *not* linear in the encoded indices.

We now explain how to overcome this issue by "opening up" the lincheck protocol into its components, described in Construction 5.6.8 in Section 5.6. The resulting protocol is summarized in Fig. 5.4.

The first message of the verifier consists of random elements η_A, η_B, η_C for the random linear combination, along with the random challenge α prescribed by Step 1 of the holographic lincheck protocol.

The subsequent two steps are straightforward to adapt due to linearity:

- In Step 2, the prover must send the evaluation of the polynomial $\hat{t}(X) := u_{\eta_A A + \eta_B B + \eta_C C}(X, \alpha)$, which by linearity equals to $\sum_{M \in \{A,B,C\}} \eta_M u_M(X, \alpha)$. The prover thus sends $t := \hat{t} \mid_L \in \mathrm{RS}[L, |H| - 1]$.
- In Step 3, the prover and verifier must run the zero knowledge sumcheck protocol (relative to a random mask r sent earlier) to show that $\sum_{b \in H} u_H(b, \alpha) \hat{f}_{\$}(b) - \hat{t}(b) \hat{f}_z(b) = 0$ where $\hat{f}_{\$}(X)$ is the low-degree extension of the vector $(\sum_{M \in \{A,B,C\}} \eta_M M) z$. By linearity, $\hat{f}_{\$}(X) = \sum_{M \in \{A,B,C\}} \eta_M \hat{f}_M(X)$, which means that the verifier can do this since the prover has sent f_z, f_A, f_B, f_C.

Then there are two steps that remain unchanged: the verifier sends the random challenge β prescribed by Step 4 of the holographic lincheck protocol, and the prover answers with the evaluation $\gamma := \hat{t}(\beta)$ prescribed in Step 5 of the holographic lincheck protocol.

The final step of the holographic lincheck protocol, Step 6, involves a rational sumcheck checking that γ is the value of the low-degree extension of $\sum_{M \in \{A,B,C\}} \eta_M M$ at (β, α). This is

the step that lacks linear structure and we need to modify it. Specifically we need to turn the expression

$$\sum_{M \in \{A,B,C\}} \eta_M \frac{v_H(\alpha)}{(\alpha - \hat{\mathsf{row}}_{\langle M^* \rangle}(X))} \cdot \frac{v_H(\beta)}{(\beta - \hat{\mathsf{col}}_{\langle M^* \rangle}(X))} \cdot \hat{\mathsf{val}}_{\langle M^* \rangle}(X)$$

into a rational function in X. This is achieved by "multiplying up" denominators, to obtain the rational function $\hat{N}(X)/\hat{D}(X)$ where

$$\hat{N}(X) := v_H(\alpha) v_H(\beta) \sum_{M \in \{A,B,C\}} \eta_M \hat{\mathsf{val}}_{\langle M^* \rangle}(X) \prod_{N \in \{A,B,C\} \setminus \{M\}} (\alpha - \hat{\mathsf{row}}_{\langle N^* \rangle}(X))(\beta - \hat{\mathsf{col}}_{\langle N^* \rangle}(X)) \tag{5.3}$$

$$\hat{D}(X) := \prod_{M \in \{A,B,C\}} (\alpha - \hat{\mathsf{row}}_{\langle M^* \rangle}(X))(\beta - \hat{\mathsf{col}}_{\langle M^* \rangle}(X)) \ . \tag{5.4}$$

Crucially, the verifier can easily evaluate $\hat{N}$ and $\hat{D}$ at any point on L by having oracle access to the encoded indices $(\mathbb{I}_A, \mathbb{I}_B, \mathbb{I}_C)$.

5.8 Holographic IOP for R1CS

We construct an efficient holographic IOP for rank-1 constraint satisfiability (R1CS). Our preprocessing zkSNARK is obtained by applying our compiler to this protocol (reparametrized to reduce soundness error).

Theorem 5.8.1. *There exists a public-coin holographic IOP* $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ *for the indexed relation* $\mathcal{R}_{\mathrm{R1CS}}$ *(Definition 5.3.2) that is a zero knowledge proof of knowledge with the following efficiency features.*

- Indexing. *The indexer* $\mathbf{I}$*, given an index* $\mathbb{i} = (\mathbb{F}, H, K, \langle A \rangle, \langle B \rangle, \langle C \rangle)$*, where* H, K *are subgroups of* $\mathbb{F}$*, runs in time* $O_{\mathbb{F}}(|K| \log |K|)$ *to compute an encoded index* $\mathbf{I}(\mathbb{i})$ *of size* $O_{\mathbb{F}}(|K|)$*. Note that* $|\mathbf{I}(\mathbb{i})| = O(|\mathbb{i}|)$.
- Proving and verification. *To achieve zero knowledge against* b *queries, the prover* $\mathbf{P}$ *and verifier* $\mathbf{V}$ *interact over* $O(\log(|K| + \mathsf{b}))$ *rounds with a round-by-round soundness (and knowledge) error of* $O((|K| + \mathsf{b})/|\mathbb{F}| + \epsilon_{\mathrm{FRI}}(\mathbb{F}, \rho, \delta))$ *where* $\rho, \delta = \Theta(1)$*. The prover* $\mathbf{P}$ *runs in time* $O_{\mathbb{F}}((|K| + \mathsf{b}) \log(|K| + \mathsf{b}))$*, and the total length of the proof oracles that it outputs is* $O_{\mathbb{F}}(|K| + \mathsf{b})$*. The verifier* $\mathbf{V}$ *runs in time* $O_{\mathbb{F}}(|v| + \log(|K| + \mathsf{b}))$*, and makes* $O(\log(|K| + \mathsf{b}))$ *queries to the encoded index and proof oracles.*

Above, $\epsilon_{\mathrm{FRI}}(\mathbb{F}, \rho, \delta)$ denotes the round-by-round soundness error of the FRI low-degree test [22] over the field $\mathbb{F}$ for proximity parameter δ and rate parameter ρ.

The rest of this section is organized as follows: (1) we introduce a generic theorem (Theorem 5.8.2) that allows us to "compile" an RS-encoded holographic IOP into a holographic IOP via a low-degree test; then (2) we show how to apply this theorem with the FRI low-degree

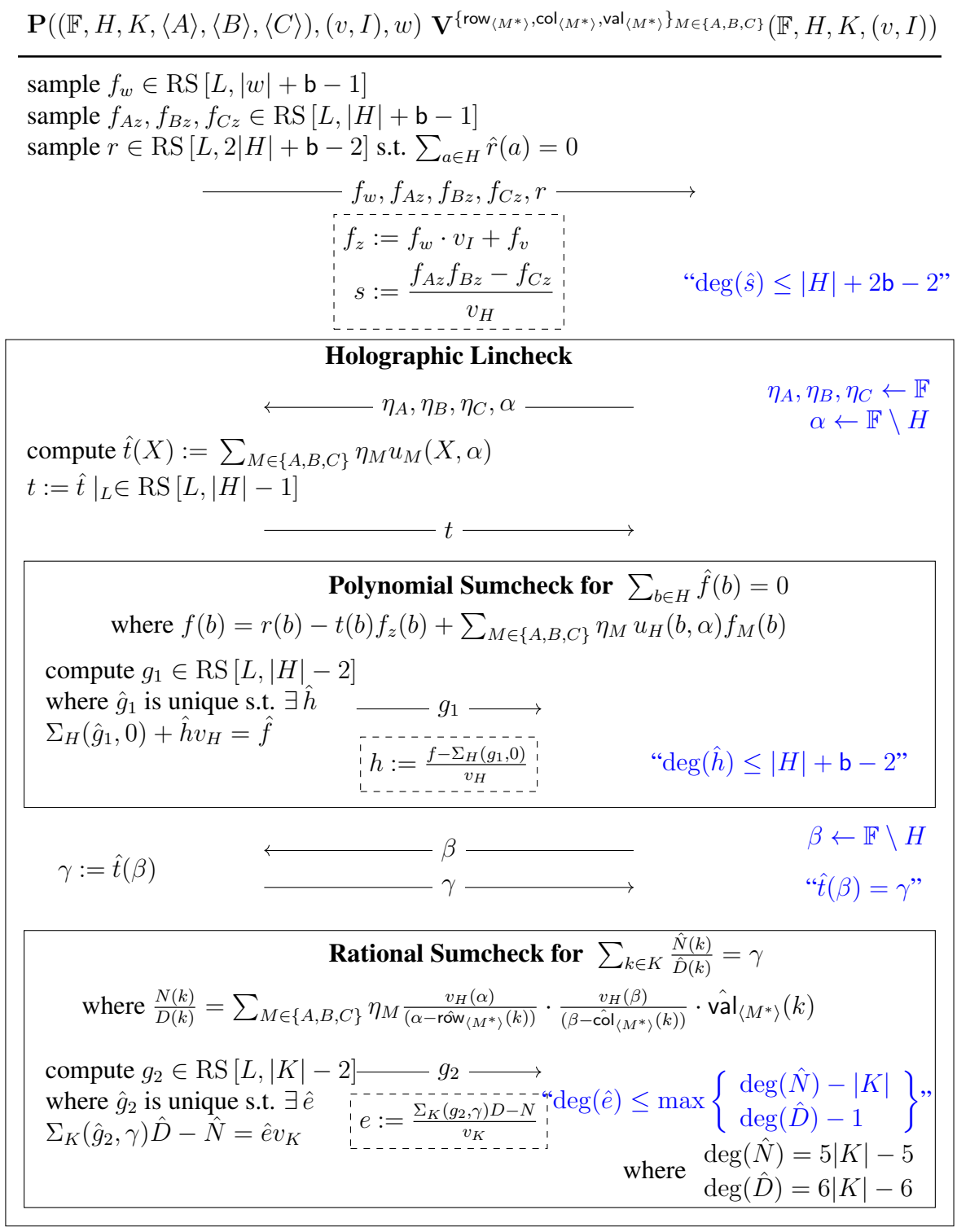


Figure 5.4: Diagram of our RS-encoded holographic IOP for R1CS (Construction 5.7.2), after applying the optimizations described in Remark 5.7.3 (which batch the three holographic linchecks into one holographic protocol).

test [22] to prove Theorem 5.8.1. Theorem 5.8.2 is adapted from Theorem 3.7.1 to handle holography and round-by-round soundness, and to more carefully account for the running time of the verifier. Before stating the theorem, we briefly describe the construction.

The compiled holographic IOP consists of two conceptual stages: first, the prover and verifier engage in the RS-encoded holographic IOP; then, the prover proves to the verifier that the oracles it sent were of degree d using the low-degree test. For efficiency, rather than proving the degree of $f_1, \ldots, f_{\mathsf{k}}$ separately, we introduce an additional round of interaction where the verifier chooses a vector $\vec{z} \in \mathbb{F}^{\mathsf{k}}$ and the prover shows that the oracle $\sum_i z_i f_i$ has degree d. (If the oracles sent have differing prescribed degrees, then we "shift" them so they all have the same degree.) Finally, for zero knowledge, the prover sends a random f_0 of degree d (before seeing $\vec{z}$) and shows instead that $f_0 + \sum_i z_i f_i$ is of degree d. We now state and prove the theorem.

Theorem 5.8.2 (adapted from Theorem 3.7.1)**.** *Suppose that we are given:*

- *an RS-encoded holographic IOP* $\mathsf{HOL}_{\mathcal{R}} = (\mathbf{I}_{\mathcal{R}}, P_{\mathcal{R}}, V_{\mathcal{R}}, \{\vec{d}_{\mathbf{I}}, \vec{d}_{\mathbf{P},1}, \ldots, \vec{d}_{\mathbf{P},\mathsf{k}}\})$ *over* L*, with maximum degree* $(d_{\mathsf{c}}, d_{\mathsf{e}})$*, for an indexed relation* $\mathcal{R}$*;*
- *a low-degree test* $(\mathbf{P}_{\mathrm{LDT}}, \mathbf{V}_{\mathrm{LDT}})$ *for the Reed–Solomon code* $\mathrm{RS}[L, d_{\mathsf{c}}]$.

Fix any proximity parameter δ^{LDT} *such that*

$$\delta^{\mathrm{LDT}} < \min\left(\frac{1-2\rho_{\mathsf{c}}}{2}, \frac{1-\rho_{\mathsf{c}}}{3}, 1-\rho_{\mathsf{e}}\right) \quad \textit{where} \quad \rho_{\mathsf{c}} := \frac{d_{\mathsf{c}}+1}{|L|} \quad \textit{and} \quad \rho_{\mathsf{e}} := \frac{d_{\mathsf{e}}+1}{|L|} \ .$$

Then we can combine the above two ingredients to obtain a holographic IOP $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *for* $\mathcal{R}$ *with the following parameters. (Parameters with superscript "*$\mathcal{R}$*" and "*LDT*" are parameters for* $(\mathbf{I}_{\mathcal{R}}, P_{\mathcal{R}}, V_{\mathcal{R}})$ *and* $(\mathbf{P}_{\mathrm{LDT}}, \mathbf{V}_{\mathrm{LDT}})$ *respectively.)*

(i) $\mathsf{k}^{\mathcal{R}} + \mathsf{k}^{\mathrm{LDT}}$ *rounds,*
(ii) query complexity $\mathsf{q}_{\pi}^{\mathrm{LDT}} + \mathsf{q}_{\mathrm{w}}^{\mathrm{LDT}} \cdot (\mathsf{k}^{\mathcal{R}} + 1)$*,*
(iii) proof length $\mathsf{L}^{\mathcal{R}} + \mathsf{L}^{\mathrm{LDT}}$*,*
(iv) soundness error $\epsilon^{\mathcal{R}} + \epsilon^{\mathrm{LDT}} + |L|/|\mathbb{F}|$*, and*
(v) **round-by-round soundness error** $\max(\epsilon_{\mathrm{rbr}}^{\mathcal{R}}, \epsilon_{\mathrm{rbr}}^{\mathrm{LDT}}, |L|/|\mathbb{F}|)$.

The new indexer $\mathbf{I}$ *equals* $\mathbf{I}_{\mathcal{R}}$*; the new prover* $\mathbf{P}$ *runs in time* $\mathrm{time}(\mathbf{P}^{\mathrm{LDT}}) + \mathrm{time}(P^{\mathcal{R}})$*; and the new verifier* $\mathbf{V}$ *runs in time* $\mathrm{time}(\mathbf{V}^{\mathrm{LDT}}) + \mathsf{q}_{\pi}^{\mathrm{LDT}} \cdot t_{\mathsf{q}}^{\mathcal{R}}$.

If $(\mathbf{I}_{\mathcal{R}}, P_{\mathcal{R}}, V_{\mathcal{R}})$ *is zero-knowledge then* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *is also zero-knowledge (with the same query bound).*

If $(\mathbf{I}_{\mathcal{R}}, P_{\mathcal{R}}, V_{\mathcal{R}})$ *has* **round-by-round knowledge error** $\kappa_{\mathrm{rbr}}^{\mathcal{R}}$ *then* $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ *has round-by-round knowledge error* $\max(\kappa_{\mathrm{rbr}}^{\mathcal{R}}, \epsilon_{\mathrm{rbr}}^{\mathrm{LDT}}, |L|/|\mathbb{F}|)$.

Proof. The proof is essentially identical to Theorem 3.7.1; note, however, that we do not need to low-degree test the encoded index since it is honestly generated.

To show round-by-round soundness, we define a state function State using the state functions $\mathsf{State}^{\mathcal{R}}$ and $\mathsf{State}^{\mathrm{LDT}}$ of the holographic IOP and low-degree test, respectively. Since the protocols are sequentially composed, we can split the transcript into three parts: $\mathsf{tr}^{\mathcal{R}}$, the first $\mathsf{k}^{\mathcal{R}}$ rounds; $\vec{z}$, the verifier message in round $\mathsf{k}^{\mathcal{R}} + 1$ (to make a full round we precede this with a "dummy" prover message); and $\mathsf{tr}^{\mathrm{LDT}}$, the last $\mathsf{k}^{\mathrm{LDT}}$ rounds. The state function is described by the following algorithm:

$\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}^{\mathcal{R}}, \vec{z}, \mathsf{tr}^{\mathrm{LDT}})$:

1. Let $(\Pi_1, m_1, \ldots, \Pi_j, m_j) := \mathsf{tr}^{\mathcal{R}}$ (for some $j \leq \mathsf{k}^{\mathcal{R}}$). If Π_i is δ^{LDT}-close to $\mathrm{RS}\left[L, \vec{d}_{\mathbf{P},i}\right]$ for all $i \in [j]$, output $\mathsf{State}^{\mathcal{R}}(\Pi'_1, m_1, \ldots, \Pi'_j, m_j)$ where $\Pi'_i \in \mathrm{RS}\left[L, \vec{d}_{\mathbf{P},i}\right]$ is the closest codeword to Π_i.
2. If $\vec{z}$ is empty then output reject; if $\vec{z}^T\Pi$ is δ^{LDT}-close to $\mathrm{RS}\left[L, d_{\mathsf{c}}\right]$ then output accept, where Π is the "stacked" proof matrix (see Protocol 3.7.2).
3. Otherwise, output $\mathsf{State}^{\mathrm{LDT}}(\mathbb{i}, \mathbb{x}, \mathsf{tr}^{\mathrm{LDT}})$.

Clearly $\mathsf{State}(\mathbb{i}, \mathbb{x}, \emptyset) = \mathsf{State}^{\mathcal{R}}(\mathbb{i}, \mathbb{x}, \emptyset) = \mathsf{reject}$. For any partial transcript tr, if tr ends during the first stage of the protocol then $\mathsf{rbr}(\mathsf{tr}) \leq \epsilon^{\mathcal{R}}_{\mathrm{rbr}}$. If tr ends with round $\mathsf{k}^{\mathcal{R}}$ and $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$ then the probability that $\vec{z}^T\Pi$ is δ^{LDT}-close to $\mathrm{RS}\left[L, d_{\mathsf{c}}\right]$ is bounded by $|L|/|\mathbb{F}|$; hence $\mathsf{rbr}(\mathsf{tr}) \leq |L|/|\mathbb{F}|$. Finally, if tr ends after round $\mathsf{k}^{\mathcal{R}} + 1$, if $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$ then $\vec{z}^T\Pi$ is δ^{LDT}-far from $\mathrm{RS}\left[L, d_{\mathsf{c}}\right]$, and so by the RBR soundness guarantee of the low-degree test $\mathsf{rbr}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) \leq \epsilon^{\mathrm{LDT}}_{\mathrm{rbr}}$.

The same state function witnesses round-by-round knowledge soundness. Suppose that for some transcript tr with $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{reject}$, $\mathsf{rbr}(\mathsf{tr}) > \max(\kappa^{\mathcal{R}}_{\mathrm{rbr}}, \epsilon^{\mathrm{LDT}}_{\mathrm{rbr}}, |L|/|\mathbb{F}|)$. Since $\mathsf{rbr}(\mathsf{tr}) > \max(\epsilon^{\mathrm{LDT}}_{\mathrm{rbr}}, |L|/|\mathbb{F}|)$, it must be that Π_i is δ^{LDT}-close to $\mathrm{RS}\left[L, \vec{d}_{\mathbf{P},i}\right]$ for all i; hence $\mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}^{\mathcal{R}}_c) = \mathsf{State}(\mathbb{i}, \mathbb{x}, \mathsf{tr}) = \mathsf{accept}$. We apply the knowledge extractor of the RS-hIOP to $\tilde{\mathbf{P}}_c$, which runs $\tilde{\mathbf{P}}$ and corrects its output words. The knowledge soundness guarantee for the RS-hIOP ensures that this extractor succeeds. □

Proof of Theorem 5.8.1. The two main ingredients in the proof are the RS-hIOP of Theorem 5.7.1 and the FRI low-degree test [22]. These are combined using Theorem 5.8.2 to build the described IOP. The indexer in our construction will choose L to be a coset of a smooth subgroup of $\mathbb{F}$ with, say, $8|K| \leq |L| \leq 16|K|$, and $(H \cup K) \cap L = \emptyset$. □

5.9 Definition of preprocessing non-interactive arguments in the ROM

We denote by $\mathcal{U}(\lambda)$ the set of all functions that map $\{0,1\}^*$ to $\{0,1\}^\lambda$. A *random oracle* with security parameter λ is a function $\rho\colon \{0,1\}^* \to \{0,1\}^\lambda$ sampled uniformly at random from $\mathcal{U}(\lambda)$.

A tuple of algorithms $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ is a *preprocessing non-interactive argument* in the random oracle model (ROM) for an indexed relation $\mathcal{R}$ if the following properties hold.

- **Completeness.** For every adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} & \rho \leftarrow \mathcal{U}(\lambda) \\ \vee & (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}^{\rho} \\ \mathcal{V}^{\rho}(\mathsf{ivk}, \mathbb{x}, \pi) = 1 & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathbb{i}) \\ & \pi \leftarrow \mathcal{P}^{\rho}(\mathsf{ipk}, \mathbb{x}, \mathbb{w}) \end{array}\right] = 1 \ .$$

- **Soundness.** For every t-query adversary $\tilde{\mathcal{P}}$,

$$\Pr\left[\begin{array}{c|c} (\mathbb{i},\mathbb{x}) \notin \mathcal{L}(\mathcal{R}) & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & (\mathbb{i},\mathbb{x},\pi) \leftarrow \tilde{\mathcal{P}}^{\rho} \\ \mathcal{V}^{\rho}(\mathsf{ivk},\mathbb{x},\pi)=1 & (\mathsf{ipk},\mathsf{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathbb{i}) \end{array}\right] \leq \epsilon(t,\lambda) \ .$$

The above formulation of completeness allows $(\mathbb{i},\mathbb{x},\mathbb{w})$ to depend on the random oracle ρ, and the above formulation of soundness allows $(\mathbb{i},\mathbb{x})$ to depend on the random oracle ρ.

All constructions in this paper achieve the stronger property of *knowledge soundness*, and optionally also the property of (statistical) *zero knowledge*. We define both of these properties below.

Knowledge soundness. We say that $\mathsf{ARG} = (\mathcal{I},\mathcal{P},\mathcal{V})$ has (adaptive) knowledge error κ if there exists an efficient extractor $\mathcal{E}$ such that for every t-query adversary $\tilde{\mathcal{P}}$ and predicate p,

$$\begin{aligned} &\Pr\left[\begin{array}{c|c} p(\vec{\mathbb{i}},\vec{\mathbb{x}},\mathsf{aux})=1 & \\ \wedge\, \forall\, j \in [\ell]\,, & (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\mathbb{w}},\mathsf{aux}) \leftarrow \mathcal{E}^{\tilde{\mathcal{P}}}(1^t,1^\lambda) \\ (\mathbb{i}_j,\mathbb{x}_j,\mathbb{w}_j) \in \mathcal{R} & \end{array}\right] \\ &\geq \Pr\left[\begin{array}{c|c} p(\vec{\mathbb{i}},\vec{\mathbb{x}},\mathsf{aux})=1 & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge\, \forall\, j \in [\ell]\,, & (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux}) \leftarrow \tilde{\mathcal{P}}^{\rho} \\ \mathcal{V}^{\rho}(\mathsf{ivk}_j,\mathbb{x}_j,\pi_j)=1 & \forall\, j\,,\ (\mathsf{ipk}_j,\mathsf{ivk}_j) \leftarrow \mathcal{I}^{\rho}(\mathbb{i}_j) \end{array}\right] - \kappa(t,\lambda,\ell) \ . \end{aligned} \tag{5.5}$$

This implies that the distributions of $(\vec{\mathbb{i}},\vec{\mathbb{x}},\mathsf{aux})$ in the following experiments are κ-close:

- $(\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux}) \leftarrow \tilde{\mathcal{P}}^{\rho}$, restricted to the space where for all j, π_j is a valid proof for $(\mathbb{i}_j,\mathbb{x}_j)$; and
- $(\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\mathbb{w}},\mathsf{aux}) \leftarrow \mathcal{E}^{\tilde{\mathcal{P}}}(1^t,1^\lambda)$, restricted to the space where for all j, $(\mathbb{i}_j,\mathbb{x}_j,\mathbb{w}_j) \in \mathcal{R}$.

Zero knowledge. We say that $\mathsf{ARG} = (\mathcal{I},\mathcal{P},\mathcal{V})$ has (adaptive statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for every t-query honest adversary $\mathcal{A}$ the distributions below are statistically close (as a function of λ):

$$\left\{(\rho,\mathbb{i},\mathbb{x},\pi) \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\mathbb{i},\mathbb{x},\mathbb{w}) \leftarrow \mathcal{A}^{\rho} \\ (\mathsf{ipk},\mathsf{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathbb{i}) \\ \pi \leftarrow \mathcal{P}^{\rho}(\mathsf{ipk},\mathbb{x},\mathbb{w}) \end{array}\right\} \quad \text{and} \quad \left\{(\rho[\mu],\mathbb{i},\mathbb{x},\pi) \;\middle|\; \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\mathbb{i},\mathbb{x},\mathbb{w}) \leftarrow \mathcal{A}^{\rho} \\ (\mu,\pi) \leftarrow \mathcal{S}^{\rho}(\mathbb{i},\mathbb{x}) \end{array}\right\} \ .$$

Above, $\rho[\mu]$ is the function that, on input x, equals $\mu(x)$ if μ is defined on x, or $\rho(x)$ otherwise. This definition uses explicitly-programmable random oracles [20]. (Non-interactive zero knowledge with non-programmable random oracles is impossible for non-trivial languages [125, 32].) Here, by an *honest adversary* we mean an adversary whose output satisfies $(\mathbb{i},\mathbb{x},\mathbb{w}) \in \mathcal{R}$ with probability 1.

Post-quantum security. The above definitions consider security against *classical* adversaries that make a bounded number of queries to the oracle (and are otherwise computationally unbounded). We also consider security against *quantum* adversaries, whose queries to the oracle can be in superposition. This setting is known as the quantum random oracle model (QROM) [48], and is the established model to study post-quantum security for constructions that use random oracles. The soundness definition and knowledge soundness definition for post-quantum security

are identical to the ones above, except that $\tilde{\mathcal{P}}^{\rho}$ is now taken to mean that $\tilde{\mathcal{P}}$ has superposition query access to ρ; the zero knowledge definition remains unchanged because indistinguishability holds against unbounded adversaries that see the whole oracle.

We do not know if, in the quantum setting, knowledge soundness with auxiliary output is polynomially related to knowledge soundness without auxiliary output.

5.10 From holographic IOPs to preprocessing arguments

We describe how to transform any public-coin holographic IOP (Section 5.4) into a corresponding preprocessing non-interactive argument in the ROM (Section 5.9). We additionally explain how the same transformation achieves post-quantum security in the QROM. A main goal is achieving adaptive knowledge soundness.

Theorem 5.10.1. *There exists a polynomial-time transformation* T *such that if* $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ *is a public-coin holographic IOP for an indexed relation* $\mathcal{R}$ *then* $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V}) := \mathrm{T}(\mathsf{HOL})$ *is a preprocessing non-interactive argument in the ROM for* $\mathcal{R}$*. The transformation* T *satisfies the following properties:*

- EFFICIENCY: *If* HOL *has oracle length* L *and query complexity* q *then* ARG *has argument size* $O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{L})$*; moreover, the time complexities of the argument indexer, prover, and verifier are as follows*

$$\begin{aligned} \mathrm{time}(\mathcal{I}) &= \mathrm{time}(\mathbf{I}) + O(\lambda \mathsf{L}) \ , \\ \mathrm{time}(\mathcal{P}) &= \mathrm{time}(\mathbf{P}) + O(\lambda \mathsf{L}) \ , \\ \mathrm{time}(\mathcal{V}) &= \mathrm{time}(\mathbf{V}) + O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{L}) \ . \end{aligned}$$

- ZK PRESERVATION: *if* HOL *is honest-verifier zero knowledge then* ARG *is adaptive statistical zero knowledge.*

- ADAPTIVE KNOWLEDGE FROM SR: *If* HOL *has state-restoration knowledge error* $\kappa_{\mathrm{sr}}(t)$ *then* ARG *has adaptive knowledge error* $\kappa(t, \lambda, \ell) = t \cdot (\kappa_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda}))$.[13]

- ADAPTIVE KNOWLEDGE FROM RBR: *If* HOL *has round-by-round knowledge error* κ_{rbr} *then* ARG *has adaptive knowledge error* $\kappa(t, \lambda, \ell) = t \cdot \kappa_{\mathrm{rbr}} + O(t^2 \cdot 2^{-\lambda})$ *in the ROM and adaptive knowledge error* $\kappa(t, \lambda, \ell) = O(t^2 \cdot \kappa_{\mathrm{rbr}} + t^3 \cdot 2^{-\lambda})$ *in the QROM.*

5.10.1 Construction

The transformation T has two parts. First, we apply the BCS transformation [32] to the holographic IOP to obtain a "holographic" non-interactive argument, namely, a non-interactive argument where the (deterministic) argument verifier is fast when given oracle access to the

[13] This is not tight; a more sophisticated analysis using state-restoration soundness directly would eliminate a factor of t. We leave this to future work, since in our application we apply the transformation to a protocol that satisfies round-by-round soundness.

encoded index. Next, we transform this into a preprocessing non-interactive argument by having the argument indexer output a Merkle commitment to the encoded index, and having the argument prover additionally output Merkle openings to the positions of the encoded index queried by the IOP verifier.

We now describe the transformation T in more detail: in Construction 5.10.2 we recall the transformation $\mathrm{T}_{\mathsf{BCS}}$ of [32], adapting its presentation to holographic IOPs (but the construction is identical otherwise); then in Construction 5.10.3 we describe the transformation T, using $\mathrm{T}_{\mathsf{BCS}}$ as a subroutine.

Construction 5.10.2 ($\mathrm{T}_{\mathsf{BCS}}$)**.** *The transformation $\mathrm{T}_{\mathsf{BCS}}$ takes as input a holographic IOP $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ and outputs the (standard) non-interactive argument $\mathsf{ARG}_{\mathsf{BCS}} = (\mathcal{P}_{\mathsf{BCS}}, \mathcal{V}_{\mathsf{BCS}})$ defined below.*

- $\mathcal{P}^{\rho}_{\mathsf{BCS}}(\mathbb{i}, \mathbb{x}, \mathbb{w})$*:*
  1. *Set $\sigma_0 := \mathsf{ivk} \| \mathbb{x}$, where $(\mathsf{ivk}, \mathsf{ipk}) \leftarrow \mathcal{I}(\mathbb{i})$ and $\mathcal{I}$ is the argument indexer in Construction 5.10.3.*
  2. *For $i = 1, \ldots, \mathsf{k}$:*
     (a) *Compute randomness $\rho_i := \rho(\sigma_{i-1})$ for the i-th round.*
     (b) *Provide ρ_i to the IOP prover $\mathbf{P}(\mathbb{i}, \mathbb{x}, \mathbb{w})$ to obtain a proof oracle Π_i.*
     (c) *Use ρ to compute a Merkle tree on Π_i, and in particular to obtain a Merkle root ω_i.*
     (d) *Set $\sigma_i := \rho(\sigma_{i-1} \| \omega_i)$.*
  3. *Compute randomness $\rho_{\mathsf{k}+1} := \rho(\sigma_{\mathsf{k}})$ for the query phase.*
  4. *Run the IOP verifier $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1})$, answering queries via the proof oracles $(\Pi_1, \ldots, \Pi_{\mathsf{k}})$, so to deduce the set of queries Q that are asked on randomness $(\rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1})$.*
  5. *Output the proof string π that contains all the Merkle roots $(\omega_1, \cdots, \omega_{\mathsf{k}})$ and, for each query in Q, an answer supported by an authentication path (against the appropriate root).*

- $\mathcal{V}^{\rho}_{\mathsf{BCS}}(\mathbb{i}, \mathbb{x}, \pi) \equiv \mathcal{V}^{\rho, \mathbf{I}(\mathbb{i})}_{\mathsf{BCS}}(\mathbb{x}, \pi)$*:*
  1. *Set $\sigma_0 := 0^{\lambda}$ and use the i-th root ω_i in π to set $\sigma_i := \rho(\sigma_{i-1} \| \omega_i)$ for $i = 1, \ldots, \mathsf{k}$.*
  2. *Compute each randomness: $\rho_1 := \rho(\sigma_0), \ldots, \rho_{\mathsf{k}} := \rho(\sigma_{\mathsf{k}-1}), \rho_{\mathsf{k}+1} := \rho(\sigma_{\mathsf{k}})$.*
  3. *Run the IOP verifier $\mathbf{V}^{\mathbf{I}(\mathbb{i})}(\mathbb{x}; \rho_1, \ldots, \rho_{\mathsf{k}}, \rho_{\mathsf{k}+1})$. Whenever $\mathbf{V}$ queries a proof oracle Π_i, validate the authentication path for this query in π, and answer the query with the corresponding value in π. (If π contains no entry for a query, reject.) Accept if and only if $\mathbf{V}$ accepts.*

We write $\mathcal{V}_{\mathsf{BCS}}$ as an algorithm with oracle access to $\mathbf{I}(\mathbb{i})$ to emphasize that $\mathrm{T}_{\mathsf{BCS}}$ is black-box with respect to $\mathbf{V}$: the queries $\mathcal{V}_{\mathsf{BCS}}$ makes to $\mathbf{I}(\mathbb{i})$ are exactly the queries $\mathbf{V}$ makes to $\mathbf{I}(\mathbb{i})$ (on appropriate randomness).

Construction 5.10.3 (T)**.** *The transformation T takes as input a public-coin holographic IOP $\mathsf{HOL} = (\mathbf{I}, \mathbf{P}, \mathbf{V})$ and outputs the preprocessing non-interactive argument $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ defined below. We let $(\mathcal{P}_{\mathsf{BCS}}, \mathcal{V}_{\mathsf{BCS}})$ be the BCS prover and verifier output by $\mathrm{T}_{\mathsf{BCS}}(\mathbf{I}, \mathbf{P}, \mathbf{V})$, and view $\mathcal{V}_{\mathsf{BCS}}$ as having oracle access to $\mathbf{I}(\mathbb{i})$.*

- Indexer. *On input* $\mathbb{i}$, $\mathcal{I}^{\rho}$ *computes the encoded index* $\mathbf{I}(\mathbb{i})$, *computes a Merkle commitment* ω *to* $\mathbf{I}(\mathbb{i})$ *using the sub-oracle* ρ_0, *and outputs the key pair* $(\mathsf{ipk}, \mathsf{ivk}) := ((\mathbb{i}, \mathbf{I}(\mathbb{i})), (\omega, \rho_0(\mathbb{i}))$.
- Prover. *On input* $(\mathsf{ipk}, \mathbb{x}, \mathbb{w})$, $\mathcal{P}^{\rho}$ *parses the proving key* ipk *as* $(\mathbb{i}, \mathbf{I}(\mathbb{i}))$, *computes* $z := \rho_0(\rho_0(\mathbb{i})\|\mathbb{x})$, *computes the output of the BCS prover* $\pi_{\mathsf{BCS}} := \mathcal{P}_{\mathsf{BCS}}^{\rho_z}(\mathbb{i}, \mathbb{x}, \mathbb{w})$, *simulates the BCS verifier* $\mathcal{V}_{\mathsf{BCS}}^{\rho_z, \mathbf{I}(\mathbb{i})}(\mathbb{x}, \pi_{\mathsf{BCS}})$ *letting* ap_i *be the authentication path for its* i*-th query to* $\mathbf{I}(\mathbb{i})$, *and outputs the proof string* $\pi := (\pi_{\mathsf{BCS}}, (\mathsf{ap}_1, \ldots, \mathsf{ap}_k))$.
- Verifier. *On input* $(\mathsf{ivk}, \mathbb{x}, \pi)$, $\mathcal{V}^{\rho}$ *parses the proof string* π *as* $(\pi_{\mathsf{BCS}}, (\mathsf{ap}_1, \ldots, \mathsf{ap}_k))$ *and* ivk *as* (ω, a), *computes* $z := \rho_0(a\|\mathbb{x})$, *and runs the BCS verifier* $\mathcal{V}_{\mathsf{BCS}}^{\rho_z, \bullet}(\mathbb{x}, \pi_{\mathsf{BCS}})$ *and answers its* i*-th query to the second oracle (denoted via the symbol "*$\bullet$*") using the provided authentication paths. If for any* i, ap_i *is not a valid authentication path with respect to* ρ_0, *the Merkle root* ω, *and the position requested in the* i*-th query, then* $\mathcal{V}$ *rejects. Otherwise,* $\mathcal{V}$ *accepts if* $\mathcal{V}_{\mathsf{BCS}}$ *does.*

Above we use certain domain separations for the random oracle. We define $\rho_b(m) := \rho(b\|m)$ *for* $b \in \{0, 1\}$ *and* $\rho_z(m) := \rho_1(z\|m)$. *The sub-oracle* ρ_0 *is used to commit to the index* $\mathbb{i}$, *while the sub-oracle* ρ_1 *is used by BCS prover and BCS verifier (further specialized with session identifier* z*).*

5.10.2 Completeness, efficiency, and non-adaptive zero knowledge

Completeness. This is straightforward from the protocol description.

Efficiency. The proof string π output by the argument prover $\mathcal{P}$ has two components: the proof string π_{BCS} output by the BCS prover $\mathcal{P}_{\mathsf{BCS}}$, and authentication paths $(\mathsf{ap}_1, \ldots, \mathsf{ap}_k)$ that answer queries by the IOP verifier $\mathbf{V}$ to the encoded index. Each of these components has size $O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{L})$. We now discuss time complexities. The overhead of the argument indexer $\mathcal{I}$ with respect to the IOP indexer $\mathbf{I}$ is $O(\lambda \cdot \mathsf{L})$, due to the cost of committing to the encoded index output by $\mathbf{I}$. The overhead of the argument prover $\mathcal{P}$ with respect to the IOP prover $\mathbf{P}$ is $O(\lambda \cdot \mathsf{L})$, due to the cost of committing to each oracle output by $\mathbf{P}$ (and the cost to answer queries by the IOP verifier $\mathbf{V}$ to the oracles or the encoded index). The overhead of the argument verifier $\mathcal{V}$ with respect to the IOP verifier $\mathbf{V}$ is $O(\lambda \cdot \mathsf{q} \cdot \log \mathsf{L})$, due to the cost to validate the authentication path associated to each query made by $\mathbf{V}$ (to a proof oracle or the encoded index).

Non-adaptive zero knowledge. The fact that the transformation T preserves zero knowledge follows from the fact that the BCS transformation $\mathrm{T}_{\mathsf{BCS}}$ preserves zero knowledge (if leaves in the Merkle tree are suitably salted), because the simulator is given the index $\mathbb{i}$ as input. See [32] for details on why if HOL is honest-verifier zero knowledge (when viewed as a non-holographic proof system) then $(\mathcal{P}_{\mathsf{BCS}}, \mathcal{V}_{\mathsf{BCS}})$ is statistical zero knowledge, in the *non-adaptive* case (where the index and instance are fixed in advance). Note that in the non-adaptive case there is no difference between classical and post-quantum statistical zero knowledge for a non-interactive protocol.

5.10.3 Non-adaptive soundness and knowledge

We first consider *non-adaptive* soundness and knowledge soundness in both the classical and post-quantum settings as a warm up to the adaptive case.

Classical soundness. Consider an index-instance pair $(\mathbb{i}, \mathbb{x})$ that is not in $\mathcal{L}(\mathcal{R})$ and a t-query malicious prover $\tilde{\mathcal{P}}$. Let E be the event that, over a random oracle $\rho \leftarrow \mathcal{U}(\lambda)$ and letting

$(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\rho_0}(\mathbb{i})$, for the proof string $\tilde{\pi} = (\tilde{\pi}_{\mathsf{BCS}}, (\mathsf{ap}_1, \ldots, \mathsf{ap}_k))$ output by $\tilde{\mathcal{P}}^{\rho}$ there exists an authentication path ap_i for some query location $j \in |\mathbf{I}(\mathbb{i})|$ that is valid with respect to ivk, ρ but the opened value is not equal to $\mathbf{I}(\mathbb{i})_j$. If E occurs then we can find a collision in ρ via $O(|\mathbf{I}(\mathbb{i})|)$ additional queries. Therefore

$$\begin{aligned} &\Pr_{\rho}[\mathcal{V}^{\rho}(\mathsf{ivk}, \mathbb{x}, \tilde{\pi}) = 1] \\ \leq &\Pr_{\rho}[\mathcal{V}^{\rho}(\mathsf{ivk}, \mathbb{x}, \tilde{\pi}) = 1 \mid \neg E] + \Pr_{\rho}[E] \\ \leq &\Pr_{z,\rho_1}[\mathcal{V}_{\mathsf{BCS}}^{\rho_z, \mathbf{I}(\mathbb{i})}(\mathbb{x}, \tilde{\pi}_{\mathsf{BCS}}) = 1] + (t + O(\mathsf{L}))^2/2^{\lambda} \ . \end{aligned} \tag{5.6}$$

The (non-adaptive) soundness guarantee of $\mathrm{T}_{\mathsf{BCS}}$ ensures that for any z, $\Pr_{\rho_1}[\mathcal{V}_{\mathsf{BCS}}^{\rho_z, \mathbf{I}(\mathbb{i})}(\mathbb{x}, \tilde{\pi}_{\mathsf{BCS}}) = 1] \leq \epsilon_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda})$, which yields the stated bound (since the query bound t can be assumed to be at least L).

Post-quantum soundness. The post-quantum soundness argument follows the same outline as the classical argument, except that: (i) we now use the result of [67] to bound the probability that the BCS verifier accepts in the QROM, and (ii) we use the quantum query lower bound for collisions [2] to bound the probability of the event E (which implies that a collision was found). This yields the stated bound.

Classical knowledge soundness. We will prove the following non-adaptive knowledge soundness property. For all $\mathbb{i}, \mathbb{x}$, and t-query adversaries $\tilde{\mathcal{P}}$,

$$\mu := \Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & (\pi; Q) \leftarrow \tilde{\mathcal{P}}^{\rho} \\ \mathcal{V}^{\rho}(\mathsf{ivk}, \mathbb{x}, \pi) = 1 & \mathbb{w} \leftarrow \mathcal{E}(\mathbb{i}, \mathbb{x}, \pi, Q) \\ & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathbb{i}) \end{array}\right] \leq \kappa_T(t, \lambda) \ .$$

where $(\pi; Q) \leftarrow \tilde{\mathcal{P}}^{\rho}$ denotes that π is the output of $\tilde{\mathcal{P}}^{\rho}$ and $Q \colon \{0,1\}^* \rightharpoonup \{0,1\}^{\lambda}$ is a database of its oracle queries and answers. Note that if $\tilde{\mathcal{P}}$ has some auxiliary output (as in the full adaptive knowledge soundness definition) then the distribution of this output is clearly preserved when we extract.

Inspection of the proof of knowledge soundness in [32] shows that if HOL is state-restoration knowledge sound in the sense of Definition 5.4.5 then the extractor $\mathcal{E}_{\mathsf{BCS}}$ for $\mathsf{ARG}_{\mathsf{BCS}}$ fulfills the guarantee

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & (\pi; Q) \leftarrow \tilde{\mathcal{P}}^{\rho} \\ \mathcal{V}^{\rho, \mathbf{I}(\mathbb{i})}(\mathbb{x}, \pi) = 1 & \mathbb{w} \leftarrow \mathcal{E}_{\mathsf{BCS}}(\mathbb{i}, \mathbb{x}, \pi, Q) \end{array}\right] \leq \kappa_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda}) \tag{5.7}$$

for all $\mathbb{i}, \mathbb{x}$ and t-query $\tilde{\mathcal{P}}$; in particular, the extractor $\mathcal{E}_{\mathsf{BCS}}$ need only see the queries and answers of the prover, and does not otherwise interact with it. This leads naturally to the extractor $\mathcal{E}$ for ARG described below:

- $\mathcal{E}(\mathbb{i}, \mathbb{x}, \pi, Q)$:
  1. Let $z := Q(0 \| Q(0 \| \mathbb{i}) \| \mathbb{x})$ be the "session identifier" for the index $\mathbb{i}$ and instance $\mathbb{x}$.

2. Let $Q' := \{(q_0, a) : (1\|z\|q_0, a) \in Q\}$ be the query-answer pairs for session identifier z.
3. Run the BCS extractor $\mathbb{w} \leftarrow \mathcal{E}_{\mathsf{BCS}}(\mathbb{i}, \mathbb{x}, \pi_{\mathsf{BCS}}, Q')$.
4. Output $\mathbb{w}$.

Fix arbitrary $\mathbb{i}, \mathbb{x}, \tilde{\mathcal{P}}$. Observe that running $\mathcal{E}$ on $\tilde{\mathcal{P}}$ is equivalent to running $\mathcal{E}_{\mathsf{BCS}}$ on the algorithm:

- $\tilde{\mathcal{P}}_0^{\rho}$:
  1. Simulate $\tilde{\mathcal{P}}$, answering its queries with uniform randomness, until it queries $(0\|\mathbb{i})$; respond with random $a \in \{0,1\}^{\lambda}$ (else abort).
  2. Continue simulating $\tilde{\mathcal{P}}$ until it queries $(0\|a\|\mathbb{x})$; respond with random $z \in \{0,1\}^{\lambda}$ (else abort).
  3. Continue simulating $\tilde{\mathcal{P}}$ until it halts and outputs $(\pi_{\mathsf{BCS}}, \vec{\mathsf{ap}})$, answering its queries $(1\|z\|q)$ with $\rho(q)$ and other queries with uniform randomness.
  4. Output π_{BCS}.

From Eq. (5.6) and standard random oracle arguments it is then straightforward to show that

$$\mu \leq \Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & (\pi_{\mathsf{BCS}}; Q) \leftarrow \tilde{\mathcal{P}}_0^{\rho} \\ \mathcal{V}_{\mathsf{BCS}}^{\rho, \mathbf{I}(\mathbb{i})}(\mathbb{x}, \pi_{\mathsf{BCS}}) = 1 & \mathbb{w} \leftarrow \mathcal{E}_{\mathsf{BCS}}(\mathbb{i}, \mathbb{x}, \pi_{\mathsf{BCS}}, Q) \end{array}\right] + O(t^2 \cdot 2^{-\lambda}) \ ,$$

which, combined with Eq. (5.7), completes the proof of classical non-adaptive knowledge soundness.

Post-quantum knowledge soundness. We omit a proof of non-adaptive post-quantum knowledge soundness. Instead, we sketch the main idea and how it differs from the classical case. Knowledge soundness for the BCS transformation in the QROM is argued in [67] by analyzing the "instability" of a certain set of partial functions related to the queries the verifier makes to the oracle. Showing non-adaptive knowledge soundness for Construction 5.10.3 amounts to showing that this quantity does not change significantly when we prefix each query with the session identifier. Note that in the classical proof above we were able to construct a new adversary $\tilde{\mathcal{P}}_0$ by simulating $\tilde{\mathcal{P}}$ and monitoring its queries to the oracle; this is not possible in the quantum setting due to no-cloning.

5.10.4 Classical adaptive knowledge from state restoration knowledge

We provide a "direct" proof of classical adaptive knowledge soundness from state-restoration knowledge of the hIOP. In some cases this provides a tighter bound in the classical setting than the bound from round-by-round soundness in Section 5.10.5; it also uses more standard techniques.

We will prove an *adaptive transcript extraction* property that, in the classical setting, directly implies the knowledge soundness property given in Section 5.9. The property states that there exists an efficient extractor $\mathcal{E}$ such that for every t-query adversary $\tilde{\mathcal{P}}$:

$$\Pr\left[\begin{array}{c|r} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} & \rho \leftarrow \mathcal{U}(\lambda) \\ \wedge & (\mathbb{i}, \mathbb{x}, \pi; Q) \leftarrow \tilde{\mathcal{P}}^{\rho} \\ \mathcal{V}^{\rho}(\mathsf{ivk}, \mathbb{x}, \pi) = 1 & \mathbb{w} \leftarrow \mathcal{E}(\mathbb{i}, \mathbb{x}, \pi, Q) \\ & (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathbb{i}) \end{array}\right] \leq \kappa_T(t, \lambda) \ .$$

In particular, if $\mathcal{E}$ satisfies the above, then we can achieve knowledge error $\kappa(t, \lambda, \ell) = \kappa_T(t, \lambda)$ in the sense of Eq. (5.5), simply by running $\mathcal{E}(\mathbb{i}_j, \mathbb{x}_j, \pi_j, Q)$ for all j.

We use a series of hybrids to prove the adaptive transcript extraction property. We assume, without loss of generality, that $\tilde{\mathcal{P}}$ repeats no oracle query. For convenience we write $\rho_b(\cdot)$ for $\rho(b\|\cdot)$ for $b \in \{0, 1\}$.

H_0**:** Real adaptive transcript extraction experiment.

1. $\rho \leftarrow \mathcal{U}(\lambda)$.
2. $(\mathbb{i}, \mathbb{x}, \pi; Q) \leftarrow \tilde{\mathcal{P}}^{\rho}$.
3. $\mathbb{w} \leftarrow \mathcal{E}(\mathbb{i}, \mathbb{x}, \pi, Q)$.
4. $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathbb{i})$.
5. If $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}$ and $\mathcal{V}^{\rho}(\mathsf{ivk}, \mathbb{x}, \pi) = 1$, output 1.

H_1**:** Choose a random query i to ρ_0 to be the commitment to the index and instance.

1. $\rho \leftarrow \mathcal{U}(\lambda)$.
2. Choose $i \in \{1, \ldots, t\}$ uniformly at random. Run $\tilde{\mathcal{P}}^{\rho}$ until just before its i-th query to ρ_0, and parse this query as $a'\|\mathbb{x}'$. Let Q_1 be the set of all queries made so far.
3. Continue running $\tilde{\mathcal{P}}^{\rho}$ until it halts and outputs $(\mathbb{i}, \mathbb{x}, \pi)$. Let Q_2 be the queries made in this phase.
4. $\mathbb{w} \leftarrow \mathcal{E}(\mathbb{i}, \mathbb{x}', \pi, Q_1 \cup Q_2)$.
5. $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}^{\rho}(\mathbb{i})$, and then parse ivk as a pair (ω, a). Recall from Construction 5.10.3 that ω is a Merkle tree commitment of $\mathbf{I}(\mathbb{i})$ and $a = \rho_0(\mathbb{i})$.
6. If $(\mathbb{i}, \mathbb{x}', \mathbb{w}) \notin \mathcal{R}$, $a' = a$, and $\mathcal{V}^{\rho}(\mathsf{ivk}, \mathbb{x}', \pi) = 1$, output 1.

H_2**:** Extract index from Q_1.

1. $\rho \leftarrow \mathcal{U}(\lambda)$.
2. Choose $i \in \{1, \ldots, t\}$ uniformly at random. Run $\tilde{\mathcal{P}}^{\rho}$ until just before its i-th query to ρ_0, and parse this query as $a'\|\mathbb{x}'$. Let Q_1 be the set of all queries made so far.
3. If there exists $((0\|q), a') \in Q_1$, then set $\mathbb{i}' := q$; otherwise, abort.
4. Continue running $\tilde{\mathcal{P}}^{\rho}$ until it halts and outputs $(\mathbb{i}, \mathbb{x}, \pi)$. Let Q_2 be the queries made in this phase.
5. $\mathbb{w} \leftarrow \mathcal{E}(\mathbb{i}', \mathbb{x}', \pi, Q_1 \cup Q_2)$.
6. $(\mathsf{ipk}', \mathsf{ivk}') \leftarrow \mathcal{I}^{\rho}(\mathbb{i}')$.
7. If $(\mathbb{i}', \mathbb{x}', \mathbb{w}) \notin \mathcal{R}$, and $\mathcal{V}^{\rho}(\mathsf{ivk}', \mathbb{x}', \pi) = 1$, output 1.

H_3**:** Unpack $\mathcal{E}$ and $\mathcal{V}$.

1. $\rho \leftarrow \mathcal{U}(\lambda)$.
2. Choose $i \in \{1, \ldots, t\}$ uniformly at random. Run $\tilde{\mathcal{P}}^{\rho}$ until just before its i-th query to ρ_0, and parse this query as $a'\|\mathbb{x}'$. Let Q_1 be the set of all queries made so far.
3. If there exists $((0\|q), a') \in Q_1$, then set $\mathbb{i}' := q$; otherwise, abort.
4. Continue running $\tilde{\mathcal{P}}^{\rho}$ until it halts and outputs $(\mathbb{i}, \mathbb{x}, \pi)$. Let Q_2 be the queries made in this phase. Parse π as $(\pi_{\mathsf{BCS}}, \vec{\mathsf{ap}})$.
5. $\mathbb{w} \leftarrow \mathcal{E}_{\mathsf{BCS}}(\mathbb{i}', \mathbb{x}', \pi_{\mathsf{BCS}}, Q'_1 \cup Q'_2)$, where $Q'_i := \{(q_0, b) : (1\|z\|q_0, a) \in Q_i\}$ and $z := Q(0\|a'\|\mathbb{x}')$.

6. If $(\mathbb{i}', \mathbb{x}', \mathbb{w}) \notin \mathcal{R}$, and $\mathcal{V}_{\mathsf{BCS}}^{\mathbf{I}(\mathbb{i}'),\rho_z}(\mathbb{x}', \pi_{\mathsf{BCS}}) = 1$, output 1.

H_4**:** Answer $\tilde{\mathcal{P}}$'s queries with uniform randomness, except for those after the i-th with the prefix $a'\|\mathbb{x}'$.

1. Choose $i \in \{1, \ldots, t\}$ uniformly at random. Run $\tilde{\mathcal{P}}$ until just before its i-th query with prefix 0, answering all of its queries with uniform randomness. Parse the i-th query as $\mathsf{ivk}\|\mathbb{x}$. Let Q_1 be the set of all (simulated) queries made so far.
2. If there exists $((0\|q), a') \in Q_1$, then set $\mathbb{i}' := q$; otherwise, abort.
3. $\rho' \leftarrow \mathcal{U}(\lambda)$.
4. Respond to the i-th query with random $z \in \{0,1\}^\lambda$. Continue running $\tilde{\mathcal{P}}$ until it halts and outputs $(\mathbb{i}, \mathbb{x}, \pi)$, answering its queries of the form $(1\|z\|q_0)$ with $\rho'(q_0)$ and all other queries uniformly at random. Let Q_2' be the set of queries made to ρ'.
5. $\mathbb{w} \leftarrow \mathcal{E}_{\mathsf{BCS}}(\mathbb{i}, \mathbb{x}, \pi_{\mathsf{BCS}}, Q_2')$.
6. If $(\mathbb{i}', \mathbb{x}', \mathbb{w}) \notin \mathcal{R}$ and $\mathcal{V}_{\mathsf{BCS}}^{\mathbf{I}(\mathbb{i}'),\rho'}(\mathbb{x}', \pi_{\mathsf{BCS}}) = 1$, output 1.

For $i = 0, \ldots, 4$, let $p_i := \Pr[H_i \text{ outputs } 1]$. Note that p_0 is the probability we wish to bound, and $p_4 \leq \kappa_{\mathrm{sr}}(t) + O(t^2 \cdot 2^{-\lambda})$ by the knowledge guarantee of the BCS transformation (Eq. (5.7)), since $\mathbb{i}'$ and $\mathbb{x}'$ are chosen independently of the oracle ρ'. We prove the following claims.

Claim 5.10.4. $p_0 \leq t \cdot p_1 + 2^{-\lambda}$.

Proof. Let E_1 be the event that $\tilde{\mathcal{P}}^\rho$ outputs $(\mathbb{i}, \mathbb{x})$ such that $0\|a\|\mathbb{x}$ is never queried for $a = \rho_0(\mathbb{i})$. Then the probability that $\mathcal{V}$ accepts given E_1 is at most $2^{-\lambda}$. If E_1 does not hold then with probability at least $1/t$, $\mathbb{x} = \mathbb{x}'$ and $a = a'$.

Claim 5.10.5. $p_1 \leq p_2 + (t+2)^2 \cdot 2^{-\lambda}$.

Proof. If there does not exist $((0\|q), a) \in Q_1$ then the probability that $a' = \rho_0(\mathbb{i}) = a$ is at most $(t+1) \cdot 2^{-\lambda}$. Otherwise, since $\rho_0(\mathbb{i}) = a = a' = \rho_0(\mathbb{i}')$, the probability that $\mathbb{i} \neq \mathbb{i}'$ is at most $(t+1)^2 \cdot 2^{-\lambda}$, since this would constitute a collision. □

Claim 5.10.6. $p_2 \leq p_3 + (t + O(\log \mathsf{L}))^2 \cdot 2^{-\lambda}$.

Proof. This follows directly from the soundness argument, and the definition of $\mathcal{E}$. □

Claim 5.10.7. $p_3 \leq p_4 + t \cdot 2^{-\lambda}$.

Proof. Let $z := \rho_0(\mathsf{ivk}\|\mathbb{x})$. Since $\tilde{\mathcal{P}}$ queries ρ_0 at $\mathsf{ivk}\|\mathbb{x}$ for the first time at query i, its queries before the i-th are independent of z. Hence the probability that any of those queries has prefix $1\|z$ is at most $t \cdot 2^{-\lambda}$. If none of these queries have prefix $1\|z$, then $Q_1' = \emptyset$, and $\mathcal{V}_{\mathsf{BCS}}$'s queries are disjoint from Q_1'. Neither $\mathcal{E}_{\mathsf{BCS}}$ nor $\mathcal{V}_{\mathsf{BCS}}$ query ρ_0, and so making the specified modifications to the oracles does not change their behavior. □

We thus obtain that $p_0 \leq t \cdot \kappa_{\mathrm{sr}}(t) + O(t^3 \cdot 2^{-\lambda})$, from which the stated expression follows. □

5.10.5 Adaptive knowledge from round-by-round knowledge

In this section we obtain a bound on adaptive knowledge soundness from round-by-round soundness. We rely heavily on concepts and notation introduced in [67, Sections 3–5], and we strongly recommend that the interested reader become familiar with these before proceeding with this section.

We begin with some additional preliminaries on modeling the random oracle. In this analysis, we will model the random oracle as a random function with fixed input length $3\lambda + 1$ and output length λ. For larger inputs to ρ, which for us will have prefix 0, we recursively define $\rho(0\|s_0\|s_1) := \rho(0\|\rho(0\|s_0)\|s_1)$ for $s_0 \in \{0,1\}^{3\lambda}$ and $s_1 \in \{0,1\}^{2k\lambda}$, $k \geq 1$ (we will implicitly pad inputs up to an odd multiple of λ). For a database $D\colon \{0,1\}^{2\lambda+1} \rightharpoonup \{0,1\}^{\lambda}$ we use almost the same convention, except when $D(b\|s_0)$ is undefined:

$$D(0\|s_0\|s_1) := \begin{cases} \perp & \text{if } D(0\|s_0) = \perp; \\ D(0\|D(0\|s_0)\|s_1) & \text{otherwise.} \end{cases}$$

For database $D\colon \{0,1\}^{3\lambda+1} \rightharpoonup \{0,1\}^{\lambda}$, let $S_b(D) := \{s_0 \in \{0,1\}^{\lambda} : \exists s_1 \in \{0,1\}^{2\lambda}, (b\|s_0\|s_1) \in \mathrm{supp}(D)\}$. We make use of the following simple lemma.

Lemma 5.10.8. *Let $D\colon \{0,1\}^{3\lambda+1} \rightharpoonup \{0,1\}^{\lambda}$ be a database. Let $k \geq 3$ be an odd integer, and let $s \in \{0,1\}^{k\lambda}, x \in \{0,1\}^{3\lambda+1} \setminus \mathrm{supp}(D), y, a \in \{0,1\}^{\lambda}$ be such that $D(0\|s) \neq (D + [x \mapsto y])(0\|s) = a$. Then $y \in \{a\} \cup S_0(D)$.*

Proof. We proceed by induction on k. Let $D' := D + [x \mapsto y]$. Note first that if $D(0\|s) \neq D'(0\|s)$, it must be that $D(0\|s) = \perp$. For the base case, if $k = 3$, then $x = 0\|s$ and so $y = a$. Now suppose that for all odd $k' < k$, the lemma holds for all $s' \in \{0,1\}^{k\lambda}$. Let $s_0\|s_1 := s$, where $s_0 \in \{0,1\}^{3\lambda}$, $s_1 \in \{0,1\}^{(k-3)\lambda}$. If $D(0\|s_0) = \perp$, then $x = 0\|s_0$, and so $D'(0\|s) = D'(0\|y\|s_1) \neq \perp$. If $k - 3 = 2$, then $y \in S_0(D)$ immediately; otherwise, $D'(0\|y\|s_1) = D'(0\|D'(0\|y\|s_1')\|s_2)$ for some $s_1' \in \{0,1\}^{2\lambda}$, and so $D'(0\|y\|s_1') \neq \perp$, whence $y \in S_0(D)$. Otherwise, let $b := D(0\|s_0)$; we have that $D(0\|b\|s_1) = \perp$, but $D'(0\|b\|s_1) = a$. Since $b\|s_1 \in \{0,1\}^{(k-2)\lambda}$, by induction $y \in \{a\} \cup S_0(D)$. □

For a database D, $z \in \{0,1\}^{\lambda}$, we write $D_{1\|z}$ for the database $\{(x,y) : (1\|z\|x, y) \in D\}$.

Lemma 5.10.9. *Let $\{\mathcal{P}_{\mathbb{i},\mathbb{x}}\}_{\mathbb{i},\mathbb{x}}$ be a set of database properties indexed by $\mathbb{i}, \mathbb{x}$. Let $\mathcal{P}$ be the property consisting of databases D for which there exist $\mathbb{i}, \mathbb{x}, a, b$ such that $D(0\|\mathbb{i}) = a$, $D(0\|a\|\mathbb{x}) = b$ and $D_{1\|b} \in \mathcal{P}_{\mathbb{i},\mathbb{x}}$. Then $I(\mathcal{P}|\bar{\mathcal{P}}_{\mathrm{col}}, t) \leq \max(\max_{\mathbb{i},\mathbb{x}} I(\mathcal{P}_{\mathbb{i},\mathbb{x}}|\bar{\mathcal{P}}_{\mathrm{col}}, t), t/2^{\lambda})$.*

Proof. We first bound $\mathrm{flip}(\bar{\mathcal{P}} \cap \bar{\mathcal{P}}_{\mathrm{col}} \to \mathcal{P} \cap \bar{\mathcal{P}}_{\mathrm{col}})$. Let $D \in \bar{\mathcal{P}} \cap \bar{\mathcal{P}}_{\mathrm{col}}$, fix some $x \in \{0,1\}^{3\lambda+1} \setminus \mathrm{supp}(D)$. We consider two cases.

- The first bit of x is 0. We show that if y is such that $D' = D + [x \mapsto y] \in \mathcal{P} \cap \bar{\mathcal{P}}_{\mathrm{col}}$, then $y \in S_0(D) \cup S_1(D)$, and so $\Pr_y[D + [x \mapsto y] \in \mathcal{P} \cap \bar{\mathcal{P}}_{\mathrm{col}}] \leq |S_0(D) \cup S_1(D)|/2^{\lambda} \leq t/2^{\lambda}$.

  By definition there exist $\mathbb{i}, \mathbb{x}, a, b$ such that $D'(0\|\mathbb{i}) = a$, $D'(0\|a\|\mathbb{x}) = b$ and $D'_{1\|b} \in \mathcal{P}_{\mathbb{i},\mathbb{x}}$. Since $D \in \bar{\mathcal{P}} \cap \bar{\mathcal{P}}_{\mathrm{col}}$ and the first bit of x is 0, one of the first two conditions does not hold for D; we consider these in turn.

- $D(0\|a\|\mathbb{x}) \neq b$; then $y \in S_0(D) \cup \{b\}$ by Lemma 5.10.8. Then since $D'_{1\|b} \in \mathcal{P}_{\mathbb{i},\mathbb{x}}$, $b \in S_1(D)$, and so $y \in S_0(D) \cup S_1(D)$.
- $D(0\|a\|\mathbb{x}) = b$ but $D(0\|\mathbb{i}) \neq a$; then $y \in S_0(D) \cup \{a\}$ by Lemma 5.10.8. Then since $D(0\|a\|\mathbb{x}) \neq \bot$, $y \in S_0(D)$.

- The first bit of x is 1. Then $D_{1\|b} \notin \mathcal{P}_{\mathbb{i},\mathbb{x}}$; let $x = 1\|b\|x'$ for some $x' \in \{0,1\}^\lambda$. Note that since $D \in \bar{\mathcal{P}}_{\text{col}}$, there exists a unique choice of $\mathbb{i}, \mathbb{x}$ such that $D(0\|D(0\|\mathbb{i})\|\mathbb{x}) = b$. Hence in this case, $\Pr_y[D + [x \mapsto y] \in \mathcal{P} \cap \bar{\mathcal{P}}_{\text{col}}] = \Pr_y[D_{1\|b} + [x' \mapsto y] \in \mathcal{P}_{\mathbb{i},\mathbb{x}} \cap \bar{\mathcal{P}}_{\text{col}}] \leq I(\mathcal{P}_{\mathbb{i},\mathbb{x}}|\bar{\mathcal{P}}_{\text{col}}, t) \leq \max_{\mathbb{i},\mathbb{x}} I(\mathcal{P}_{\mathbb{i},\mathbb{x}}|\bar{\mathcal{P}}_{\text{col}}, t)$.

We now bound $\text{flip}(\mathcal{P} \cap \bar{\mathcal{P}}_{\text{col}} \to \bar{\mathcal{P}} \cap \bar{\mathcal{P}}_{\text{col}})$. Let $D \in \mathcal{P} \cap \bar{\mathcal{P}}_{\text{col}}$. Note that if the first bit of x is 0, there is no y such that $D + [x \mapsto y] \in \bar{\mathcal{P}} \cap \bar{\mathcal{P}}_{\text{col}}$. If the first bit of x is 1, $\Pr_y[D + [x \mapsto y] \in \bar{\mathcal{P}} \cap \bar{\mathcal{P}}_{\text{col}}] \leq \max_{\mathbb{i},\mathbb{x}} I(\mathcal{P}_{\mathbb{i},\mathbb{x}}|\bar{\mathcal{P}}_{\text{col}}, t)$ by a similar argument to the above. This completes the proof. □

To simplify the subsequent analysis we define a modified verifier algorithm $\mathcal{V}_0(\mathbb{i}, \mathbb{x}, \pi)$ which parses π as $(\pi_{\mathsf{BCS}}, \vec{\mathsf{ap}})$, computes $z := \rho_0(\rho_0(\mathbb{i})\|\mathbb{x})$ and accepts if $\mathcal{V}_{\mathsf{BCS}}^{\rho_1(z\|\cdot),\mathbf{I}(\mathbb{i})}(\mathbb{x}, \pi_{\mathsf{BCS}})$ accepts. Fix some predicate p. We denote by μ the probability

$$\Pr\left[\begin{array}{c} p(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{aux}) = 1 \\ \wedge \forall\, j \in [\ell]\,, \\ \mathcal{V}_0^\rho(\mathbb{i}_j, \mathbb{x}_j, \pi_j) = 1 \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) \leftarrow \tilde{\mathcal{P}}^\rho \end{array}\right] .$$

By a similar argument to the non-adaptive soundness case,

$$\mu \geq \Pr\left[\begin{array}{c} p(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \mathsf{aux}) = 1 \\ \wedge \forall\, j \in [\ell]\,, \\ \mathcal{V}^\rho(\mathsf{ivk}_j, \mathbb{x}_j, \pi_j) = 1 \end{array} \middle| \begin{array}{r} \rho \leftarrow \mathcal{U}(\lambda) \\ (\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) \leftarrow \tilde{\mathcal{P}}^\rho \\ \forall\, j\,,\ (\mathsf{ipk}_j, \mathsf{ivk}_j) \leftarrow \mathcal{I}^\rho(\mathbb{i}_j) \end{array}\right] - O(t^3 \cdot 2^{-\lambda}) \ ,$$

and so it suffices to show the proof of knowledge property for the verifier $\mathcal{V}_0$.

Let $\mathcal{P}_{\mathcal{V}(\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi})}$ be the set of databases Q such that the verifier accepts: i.e., Q such that for all $j \in [\ell]$, $\mathcal{V}_0^Q(\mathbb{i}_j, \mathbb{x}_j, \pi_j) = 1$. Let $\mathcal{P}_{\mathcal{E}(\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi})}$ be the set of databases Q such that the extractor succeeds: i.e., Q such that for all $j \in [\ell]$, $\mathbb{w}_j := \mathcal{E}(\mathbb{i}_j, \mathbb{x}_j, \pi_j, Q)$ is such that $(\mathbb{i}_j, \mathbb{x}_j, \mathbb{w}_j) \in \mathcal{R}$. Finally, let $\mathcal{P}_{\mathcal{V}\setminus\mathcal{E}}$ be the set of databases Q where the verifier accepts but the extractor fails for some choice of $(\mathbb{i}, \mathbb{x}, \pi)$: i.e., Q such that there exists $(\mathbb{i}, \mathbb{x}, \pi)$ for which $\mathcal{V}_0^Q(\mathbb{i}, \mathbb{x}, \pi) = 1$ but $\mathbb{w} := \mathcal{E}(\mathbb{i}, \mathbb{x}, \pi, Q)$ is such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R}$.

We lower bound the success probability of the extractor in the quantum setting in terms of

these properties as follows. The classical proof is analogous and so we will not give it explicitly.

$$\begin{aligned}
&\Pr\left[\begin{array}{c|c} p(\vec{\mathbb{i}},\vec{\mathbb{x}},\mathsf{aux})=1 \\ \wedge\,\forall\, j\in[\ell]\,, \\ (\mathbb{i}_j,\mathbb{x}_j,\mathbb{w}_j)\in\mathcal{R} \end{array}\;\middle|\; \begin{array}{c} (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux};Q)\leftarrow\mathsf{Sim}^*(\tilde{\mathcal{P}}) \\ \forall\, j\in[\ell]\,,\,\mathbb{w}_j\leftarrow\mathcal{E}(\mathbb{i}_j,\mathbb{x}_j,\pi_j;Q) \end{array}\right] \\
&\geq \Pr\left[\begin{array}{c} p(\vec{\mathbb{i}},\vec{\mathbb{x}},\mathsf{aux})=1 \\ \wedge\, Q\in\mathcal{P}_{\mathcal{E}(\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi})}\cap\mathcal{P}_{\mathcal{V}(\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi})} \end{array}\;\middle|\; (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux};Q)\leftarrow\mathsf{Sim}^*(\tilde{\mathcal{P}})\right] \\
&\geq \Pr\left[\begin{array}{c} p(\vec{\mathbb{i}},\vec{\mathbb{x}},\mathsf{aux})=1 \\ \wedge\, Q\in\mathcal{P}_{\mathcal{V}(\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi})} \end{array}\;\middle|\; (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux};Q)\leftarrow\mathsf{Sim}^*(\tilde{\mathcal{P}})\right] \\
&\qquad -\Pr\left[Q\in\mathcal{P}_{\mathcal{V}\setminus\mathcal{E}}\;\middle|\; (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux};Q)\leftarrow\mathsf{Sim}^*(\tilde{\mathcal{P}})\right]
\end{aligned}$$

By [152, Lemma 5] (which connects the success probability of Sim^* with a real execution), and since the verifier makes $O(\mathsf{q}\log\mathsf{L})$ queries to the oracle, the first term on the last line is at least $\mu - O(\mathsf{q}\log\mathsf{L}\cdot 2^{-\lambda})$.

By construction of $\mathcal{V}$ and $\mathcal{E}$, $\mathcal{P}_{\mathcal{V}\setminus\mathcal{E}}$ is exactly of the form described in Lemma 5.10.9 for the set $\{\mathcal{P}_{\mathbb{i},\mathbb{x}}\}_{\mathbb{i},\mathbb{x}}$ where $\mathcal{P}_{\mathbb{i},\mathbb{x}}$ is the set of databases such that there exists a proof π for which $\mathcal{V}^D_{\mathsf{BCS}}(\mathbb{i},\mathbb{x},\pi)$ accepts but $\mathcal{E}_{\mathsf{BCS}}(\mathbb{i},\mathbb{x},\pi,D)$ fails to produce a valid witness. By [67, Proposition 8.14], $I(\mathcal{P}_{\mathbb{i},\mathbb{x}}|\bar{\mathcal{P}}_{\mathrm{col}},t)\leq\kappa_{\mathrm{rbr}}+O(t\cdot 2^{-\lambda})$. Hence by Lemma 5.10.9, $I(\mathcal{P}_{\mathcal{V},\mathcal{E}}|\bar{\mathcal{P}}_{\mathrm{col}},t)\leq\kappa_{\mathrm{rbr}}+O(t\cdot 2^{-\lambda})$. Then by [67, Lemma 5.13] (their "lifting lemma"),

$$\Pr\left[Q\in\mathcal{P}_{\mathcal{V}\setminus\mathcal{E}}\;\middle|\; (\mathbb{i},\mathbb{x},\pi;Q)\leftarrow\mathsf{Sim}^*(\tilde{\mathcal{P}})\right]\leq 6t^2\cdot\kappa_{\mathrm{rbr}}+O(t^3\cdot 2^{-\lambda})\ .$$

We thus obtain

$$\Pr\left[\begin{array}{c|c} p(\vec{\mathbb{i}},\vec{\mathbb{x}},\mathsf{aux})=1 \\ \wedge\,\forall\, j\in[\ell]\,, \\ (\mathbb{i}_j,\mathbb{x}_j,\mathbb{w}_j)\in\mathcal{R} \end{array}\;\middle|\; \begin{array}{c} (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux};Q)\leftarrow\mathsf{Sim}^*(\tilde{\mathcal{P}}) \\ \forall\, j\in[\ell]\,, \\ \mathbb{w}_j\leftarrow\mathcal{E}(\mathbb{i}_j,\mathbb{x}_j,\pi_j;Q) \end{array}\right]\geq\mu-6t^2\cdot\kappa_{\mathrm{rbr}}-O((t^3+\mathsf{q}\log\mathsf{L})\cdot 2^{-\lambda})\ ,$$

which completes the proof.

5.10.6 Adaptive zero knowledge

To achieve adaptive zero knowledge we need a minor but standard modification to Construction 5.10.3. In our modified construction, the prover chooses a random seed $\alpha\in\{0,1\}^\lambda$ that it prefixes to all queries to ρ_z; it then provides α in the proof. It is relatively straightforward to show that this preserves (adaptive) soundness and knowledge soundness, both in the classical and quantum settings.

In the classical setting, it is easy to show that this modification achieves adaptive zero knowledge. The simulator is identical to the non-adaptive simulator, except that it prefixes its simulated queries (and hence the locations in μ) with a random α. The probability that $\mathcal{A}$ ever queries a location with prefix α is at most $t/2^{-\lambda}$, and so the output of $\mathcal{A}$ is almost independent from oracle locations with prefix α. Zero knowledge then follows by the non-adaptive (statistical) zero knowledge guarantee.

In the quantum setting, the argument is more delicate; we only sketch it here. We rely on the strong one-way-to-hiding lemma of [8]. Using this lemma, one can show that for any fixed oracles f, g, the output distribution of any t-query adversary $\mathcal{A}^f$ is $O(t^2/2^{-\lambda})$-close to $\mathcal{A}^{f_g}$, where f_g is a random function drawn by choosing $\alpha \in \{0,1\}^\lambda$ uniformly at random and setting $f_g(\alpha \| q) = g(q)$, and $f_g(q') = f(q')$ otherwise. This again shows that the output of $\mathcal{A}$ is almost independent of any programmed oracle location, and so zero knowledge follows by the non-adaptive (statistical) zero knowledge guarantee.

5.11 Recursive composition in the URS model

We describe how to transform any preprocessing SNARK in the URS model into a preprocessing PCD scheme in the URS model. The transformation preserves post-quantum security.

This section is organized as follows. In Section 5.11.1 we define preprocessing SNARKs in the URS model. In Section 5.11.2 we define preprocessing PCD schemes in the URS model. In Section 5.11.3 we state the properties of the transformation from SNARK to PCD. In sec:recursion-efficiency we describe the construction and prove its efficiency properties. In Section 5.11.5 we prove the security properties.

In this section by "polynomial-size" we mean a (non-uniform) family of polynomial-size circuits.

5.11.1 Preprocessing non-interactive arguments (of knowledge) in the URS model

Informally, the definition of a preprocessing SNARK in the URS model is similar to the definition of a preprocessing SNARK in the random oracle model (see Section 5.9) except that the random oracle is replaced by a $\mathrm{poly}(\lambda)$-size uniform random string urs. The formal definition follows.

A tuple of algorithms $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ is a *preprocessing non-interactive argument (of knowledge)* in the uniform random string (URS) model for an indexed relation $\mathcal{R}$ if the following properties hold.

Completeness. For every adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c} (\mathbb{i}, \mathbb{x}, \mathbb{w}) \notin \mathcal{R} \\ \vee \\ \mathcal{V}(\mathsf{urs}, \mathsf{ivk}, \mathbb{x}, \pi) = 1 \end{array} \middle| \begin{array}{r} \mathsf{urs} \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ (\mathbb{i}, \mathbb{x}, \mathbb{w}) \leftarrow \mathcal{A}(\mathsf{urs}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, \mathbb{i}) \\ \pi \leftarrow \mathcal{P}(\mathsf{urs}, \mathsf{ipk}, \mathbb{x}, \mathbb{w}) \end{array}\right] = 1 \ .$$

The above formulation of completeness allows $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ to depend on the reference string urs.

Knowledge soundness. We say that $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ has (adaptive) knowledge error κ if for every polynomial-size adversary $\tilde{\mathcal{P}}$ there exists a polynomial-size extractor $\mathcal{E}_{\tilde{\mathcal{P}}}$ such that for

every predicate p,

$$\Pr\left[\begin{array}{c|c} p(\mathsf{urs},\vec{\mathbb{i}},\vec{\mathbb{x}},\mathsf{aux})=1 & \mathsf{urs}\leftarrow\{0,1\}^{\mathrm{poly}(\lambda)} \\ \wedge\forall\, j\in[\ell]\,, & (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\mathbb{w}},\mathsf{aux})\leftarrow\mathcal{E}_{\tilde{\mathcal{P}}}(\mathsf{urs}) \\ (\mathbb{i}_j,\mathbb{x}_j,\mathbb{w}_j)\in\mathcal{R} & \end{array}\right]$$

$$\geq \Pr\left[\begin{array}{c|c} p(\mathsf{urs},\vec{\mathbb{i}},\vec{\mathbb{x}},\mathsf{aux})=1 & \mathsf{urs}\leftarrow\{0,1\}^{\mathrm{poly}(\lambda)} \\ \wedge\forall\, j\in[\ell]\,, & (\vec{\mathbb{i}},\vec{\mathbb{x}},\vec{\pi},\mathsf{aux})\leftarrow\tilde{\mathcal{P}}(\mathsf{urs}) \\ \mathcal{V}(\mathsf{ivk}_j,\mathbb{x}_j,\pi_j)=1 & \forall\, j\,,\ (\mathsf{ipk}_j,\mathsf{ivk}_j)\leftarrow\mathcal{I}(\mathsf{urs},\mathbb{i}_j) \end{array}\right] - \kappa(\lambda,\ell)\ . \tag{5.8}$$

Zero knowledge. We say that $\mathsf{ARG}=(\mathcal{I},\mathcal{P},\mathcal{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that for every polynomial-size honest adversary $\mathcal{A}$,

$$\left\{(\mathsf{urs},\mathbb{i},\mathbb{x},\pi)\;\middle|\;\begin{array}{r}\mathsf{urs}\leftarrow\{0,1\}^{\mathrm{poly}(\lambda)}\\(\mathbb{i},\mathbb{x},\mathbb{w})\leftarrow\mathcal{A}(\mathsf{urs})\\(\mathsf{ipk},\mathsf{ivk})\leftarrow\mathcal{I}(\mathsf{urs},\mathbb{i})\\\pi\leftarrow\mathcal{P}(\mathsf{urs},\mathsf{ipk},\mathbb{x},\mathbb{w})\end{array}\right\}\quad\text{and}\quad\left\{(\mathsf{urs},\mathbb{i},\mathbb{x},\pi)\;\middle|\;\begin{array}{r}(\mathsf{urs},\tau)\leftarrow\mathcal{S}(1^\lambda)\\(\mathbb{i},\mathbb{x},\mathbb{w})\leftarrow\mathcal{A}(\mathsf{urs})\\\pi\leftarrow\mathcal{S}(\mathbb{i},\mathbb{x},\tau)\end{array}\right\}\ ,$$

where here an adversary is honest if its output satisfies $(\mathbb{i},\mathbb{x},\mathbb{w})\in\mathcal{R}$ with probability 1.

Remark 5.11.1 (post-quantum security)**.** The above definitions consider security against *classical* polynomial-size adversaries. We also consider security against *quantum* polynomial-size adversaries. The definitions for this case are identical, except that $\tilde{\mathcal{P}}$ is a (non-uniform) family of polynomial-size *quantum* circuits (as is the zero knowledge adversary $\mathcal{A}$).

5.11.2 Preprocessing PCD in the URS model

We have informally introduced PCD in Section 5.2.5. Formally, a triple of algorithms $\mathsf{PCD}=(\mathbb{I},\mathbb{P},\mathbb{V})$ is a *proof-carrying data scheme* (PCD scheme) in the uniform random string (URS) model for a class of compliance predicates F if the properties below hold.

Definition 5.11.2. *A* **transcript** T *is a directed acyclic graph where each vertex $u\in V(\mathsf{T})$ is labeled by a local data $z^{(u)}_{\mathsf{loc}}$ and each edge $e\in E(\mathsf{T})$ is labeled by a message $z^{(e)}\neq\perp$. The* **output** *of a transcript T, denoted $\mathsf{o}(\mathsf{T})$, is $z^{(e)}$ where $e=(u,v)$ is the lexicographically-first edge such that v is a sink.*

Definition 5.11.3. *A vertex $u\in V(\mathsf{T})$ is* **Φ-compliant** *for $\Phi\in\mathsf{F}$ if for all outgoing edges $e=(u,v)\in E(\mathsf{T})$:*

- *(base case) if u has no incoming edges, $\Phi(z^{(e)},z^{(u)}_{\mathsf{loc}},\perp,\ldots,\perp)$ accepts;*
- *(recursive case) if u has incoming edges $e_1,\ldots,e_m$, $\Phi(z^{(e)},z^{(u)}_{\mathsf{loc}},z^{(e_1)},\ldots,z^{(e_m)})$ accepts.*

We say that T is **Φ-compliant** *if all of its vertices are Φ-compliant.*

Completeness. For every adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|c}\Big(\Phi\in\mathsf{F}\ \wedge\ \big(\forall i,z_i=\perp\vee\forall i,\ \mathbb{V}(\mathsf{ivk},z_i,\pi_i)=1\big)\ \wedge & \mathsf{urs}\leftarrow\{0,1\}^{\mathrm{poly}(\lambda)}\\ \Phi(z,z_{\mathsf{loc}},z_1,\ldots,z_m)\text{ accepts}\Big) & (\Phi,z,z_{\mathsf{loc}},[z_i,\pi_i]_{i=1}^m)\leftarrow\mathcal{A}(\mathsf{urs})\\ \Downarrow & (\mathsf{ipk},\mathsf{ivk})\leftarrow\mathbb{I}(\mathsf{urs},\Phi)\\ \mathbb{V}(\mathsf{ivk},z,\pi)=1 & \pi\leftarrow\mathbb{P}(\mathsf{ipk},z,z_{\mathsf{loc}},[z_i,\pi_i]_{i=1}^m)\end{array}\right]=1\ .$$

Observe that this completeness condition is fairly strong: it requires that $\mathbb{P}$ produce a valid PCD proof for any valid $z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m$, even if they were not honestly (or even efficiently) generated.

Knowledge soundness. We say that $\mathsf{PCD} = (\mathbb{I}, \mathbb{P}, \mathbb{V})$ has knowledge soundness $\kappa(\lambda)$ if there exists some polynomial e such that for every polynomial-size adversary $\tilde{\mathbb{P}}$, there exists an extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ of size at most $e(|\tilde{\mathbb{P}}|)$ such that for every predicate p,

$$\begin{aligned}
&\Pr\left[\begin{array}{c|c} \Phi \in \mathsf{F} \wedge p(\mathsf{urs}, \Phi, \mathsf{o}(\mathsf{T})) = 1 \\ \wedge\, \mathsf{T} \text{ is } \Phi\text{-compliant} \end{array} \middle| \begin{array}{c} \mathsf{urs} \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ (\Phi, \mathsf{T}) \leftarrow \mathbb{E}_{\tilde{\mathbb{P}}}(\mathsf{urs}) \end{array}\right] \\
&\geq \Pr\left[\begin{array}{c|c} \Phi \in \mathsf{F} \wedge p(\mathsf{urs}, \Phi, \mathsf{o}) = 1 \\ \wedge\, \mathcal{V}(\mathsf{ivk}, \mathsf{o}, \pi) = 1 \end{array} \middle| \begin{array}{c} \mathsf{urs} \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ (\Phi, \mathsf{T}) \leftarrow \tilde{\mathbb{P}} \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathsf{urs}, \Phi) \end{array}\right] - \kappa(\lambda) \ .
\end{aligned}$$

Zero knowledge. We say that $\mathsf{PCD} = (\mathbb{I}, \mathbb{P}, \mathbb{V})$ has (statistical) zero knowledge if there exists a probabilistic polynomial-time simulator $\mathbb{S}$ such that for all honest adversaries $\mathcal{A}$ the distributions below are statistically close:

$$\left\{ (\mathsf{urs}, \Phi, z, \pi) \;\middle|\; \begin{array}{r} \mathsf{urs} \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)} \\ (\Phi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{urs}) \\ (\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathbb{I}(\mathsf{urs}, \Phi) \\ \pi \leftarrow \mathbb{P}(\mathsf{ipk}, \Phi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \end{array} \right\}$$

and

$$\left\{ (\mathsf{urs}, \Phi, z, \pi) \;\middle|\; \begin{array}{r} (\mathsf{urs}, \tau) \leftarrow \mathbb{S} \\ (\Phi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{urs}) \\ \pi \leftarrow \mathbb{S}(\Phi, z, \tau) \end{array} \right\} \ .$$

Here, an adversary is honest if its output satisfies the implicant of the completeness condition with probability 1, namely: $\Phi \in \mathsf{F}$, $\Phi(z, z_{\mathsf{loc}}, z_1, \ldots, z_m) = 1$, and either for all i, $z_i = \perp$, or for all i, $\mathbb{V}(\mathsf{ivk}, z_i, \pi_i) = 1$.

Efficiency. The indexer $\mathbb{I}$, prover $\mathbb{P}$ and verifier $\mathbb{V}$ run in polynomial time. A proof π has size $\mathrm{poly}(\lambda, |\Phi|)$; in particular, it is not permitted to grow with each application of $\mathbb{P}$. In general the indexer can be incorporated into the prover and verifier; we consider it separately since this allows the verifier to potentially run in time sublinear in $|\Phi|$.

5.11.3 Theorem statement

The key parameter that determines the efficiency of the preprocessing PCD scheme is the size of the preprocessing SNARK verifier as a circuit (or constraint system), as captured by the following definition.

Definition 5.11.4. *Let $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ be a preprocessing non-interactive argument in the URS model. We denote by $\mathcal{V}^{(\lambda,N,k)}$ the circuit (or constraint system) corresponding to the computation of the SNARK verifier $\mathcal{V}$, for security parameter λ, when checking indices of size at most N and instances of size at most k. Hence, for every $\mathsf{urs} \in \{0,1\}^{\mathrm{poly}(\lambda)}$ and index-instance pair*

$(\mathbb{i}, \mathbb{x})$ *with* $|\mathbb{i}| \leq N$ *and* $|\mathbb{x}| \leq k$*, index key pair* $(\mathsf{ipk}, \mathsf{ivk}) \in \mathcal{I}(\mathsf{urs}, \mathbb{i})$*, and candidate proof* π*, we have* $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, \mathbb{x}, \pi) = \mathcal{V}(\mathsf{urs}, \mathsf{ivk}, \mathbb{x}, \pi)$*. We denote by* $\mathsf{v}(\lambda, N, k)$ *the size of the circuit* $\mathcal{V}^{(\lambda,N,k)}$*, and by* $|\mathsf{ivk}(\lambda, N)|$ *the size of the index verification key* ivk*.*

The *depth* of a compliance predicate $\Phi\colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$, denoted $d(\Phi)$, is the maximum depth of any Φ-compliant transcript T. We prove the following theorem, which constructs a PCD system for constant-depth compliance predicates from any sufficiently efficient preprocessing SNARK.

Theorem 5.11.5. *There exists a polynomial-time transformation* T *such that if* $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ *is an adaptive preprocessing SNARK for* $\mathcal{R}_{\mathrm{R1CS}}$ *in the URS model then* $\mathsf{PCD} = (\mathbb{I}, \mathbb{P}, \mathbb{V}) := \mathrm{T}(\mathsf{ARG})$ *is a preprocessing PCD scheme in the URS model for constant-depth compliance predicates, provided*

$$\exists\, \epsilon \in (0,1) \text{ and a polynomial } \alpha \text{ s.t. } \mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) = O(N^{1-\epsilon} \cdot \alpha(\lambda, \ell)) \ .$$

Moreover, if the size of the predicate $\Phi\colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$ *is* $f = \omega((m \cdot \alpha(\lambda, \ell))^{1/\epsilon})$ *then the PCD indexer, PCD prover, and PCD verifier run in time that equal those of the SNARK indexer, SNARK prover, and SNARK verifier on R1CS indices* $\mathbb{i}$ *of size* $f + o(f)$ *(and R1CS instances* $\mathbb{x}$ *of size* $O(\lambda) + \ell$*).*

If ARG *is adaptive zero knowledge, then* PCD *is adaptive zero knowledge.*

If ARG *is secure against quantum adversaries, then* PCD *is secure against quantum adversaries.*

Remark 5.11.6. Our preprocessing zkSNARK for R1CS, FRACTAL, achieves the following verifier size:

$$\mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) = O(\lambda\ell + \lambda^2 \log^2(N)) \ ,$$

assuming a choice of cryptographic hash function that can be expressed via a constraint system of size $O(\lambda)$. This means that we may take any $\epsilon \in (0,1)$ and $\alpha(\lambda, \ell) := \lambda(\lambda + \ell)$. In particular, if the size of a compliance predicate Φ grows as $(m\lambda(\lambda + \ell))^{1+\delta}$ for any $\delta > 0$, then the time bounds in Theorem 5.11.5 hold for us.

5.11.4 Construction and its efficiency

We describe how to construct the preprocessing PCD scheme, and then prove the efficiency properties stated in Theorem 5.11.5. We defer proving the security properties to Section 5.11.5.

Construction 5.11.7 (from SNARK to PCD)**.** *Let* $\mathsf{ARG} = (\mathcal{I}, \mathcal{P}, \mathcal{V})$ *be a preprocessing SNARK for R1CS. We describe how to construct a preprocessing PCD scheme* $\mathsf{PCD} = (\mathbb{I}, \mathbb{P}, \mathbb{V})$*.*

Given a compliance predicate $\Phi\colon \mathbb{F}^{(m+2)\ell} \to \mathbb{F}$*, the circuit that realizes the recursion is as follows.*

$R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}\big((\mathsf{ivk}, z_{\mathsf{out}}), (z_{\mathsf{loc}}, (z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}})_{i\in[m]})\big)$*:*

1. *Check that the compliance predicate* $\Phi(z_{\mathsf{out}}, z_{\mathsf{loc}}, z^{(1)}_{\mathsf{in}}, \ldots, z^{(m)}_{\mathsf{in}})$ *accepts.*
2. *If there exists* i *such that* $(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}}) \neq \perp$*:*

   check that, for every $i \in [m]$*, the SNARK verifier* $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(i)}_{\mathsf{in}}), \pi^{(i)}_{\mathsf{in}})$ *accepts.*
3. *If the above checks hold, output* 0*; otherwise, output* 1*.*

Next we describe the indexer $\mathbb{I}$, prover $\mathbb{P}$, and verifier $\mathbb{V}$ of the PCD scheme.

- $\mathbb{I}(\mathsf{urs}, \Phi)$*:*
  1. *Compute $N := N(\lambda, |\Phi|, m, \ell)$, where N is as defined in Lemma 5.11.8 below.*
  2. *Construct the circuit $R := R_{\mathcal{V},\Phi,\mathsf{urs}}^{(\lambda,N,|\mathsf{ivk}(\lambda,N)|+\ell)}$.*
  3. *Compute the index key pair $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, R)$.*
- $\mathbb{P}(\mathsf{urs}, \mathsf{ipk}, z_{\mathsf{out}}, z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}})$*: output the proof $\pi_{\mathsf{out}} \leftarrow \mathcal{P}\big(\mathsf{urs}, \mathsf{ipk}, (\mathsf{ivk}, z_{\mathsf{out}}), (z_{\mathsf{loc}}, \vec{z}_{\mathsf{in}}, \vec{\pi}_{\mathsf{in}})\big)$.*
- $\mathbb{V}(\mathsf{urs}, \mathsf{ivk}, z_{\mathsf{out}}, \pi_{\mathsf{out}})$*: accept if $\mathcal{V}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z_{\mathsf{out}}), \pi_{\mathsf{out}})$ accepts.*

Proof of Theorem 5.11.5 (efficiency). Denote by f the size of Φ as an R1CS instance. In Construction 5.11.7, the explicit input consists of the index verification key ivk, whose size depends on N and λ, and a message z whose size is ℓ (independent of N). The security parameter λ is also independent of N. The circuit on which we wish to invoke $\mathcal{V}$ is of size

$$S(\lambda, f, m, \ell, N) = S_0(f, m, \ell) + m \cdot \mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) \quad \text{for some} \quad S_0(f, m, \ell) = f + O(m\ell) \ .$$

We want to find a function N such that $S(\lambda, f, m, \ell, N(\lambda, f, m, \ell)) \leq N(\lambda, f, m, \ell)$ and N is not too large.

Lemma 5.11.8. *Suppose that for every security parameter $\lambda \in \mathbb{N}$ and message size $\ell \in \mathbb{N}$ the ratio of verifier circuit size to index size $\mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell)/N$ is monotone decreasing in N. Then there exists a size function $N(\lambda, f, m, \ell)$ such that*

$$\forall\, \lambda, f, m, \ell \in \mathbb{N} \quad S(\lambda, f, m, \ell, N(\lambda, f, m, \ell)) \leq N(\lambda, f, m, \ell) \ .$$

Moreover if for some $\epsilon > 0$ and some increasing function α it holds that, for all N, λ, ℓ sufficiently large,

$$\mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) \leq N^{1-\epsilon}\alpha(\lambda, \ell)$$

then, for all λ, ℓ sufficiently large,

$$N(\lambda, f, m, \ell) \leq O(f) + (2m \cdot \alpha(\lambda, \ell))^{1/\epsilon} \ .$$

Proof. Let $N_0 := N_0(\lambda, m, \ell)$ be the smallest integer such that $\mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell)/N < 1/(2m)$; this exists because of the monotone decreasing condition. Let $N(\lambda, f, m, \ell) := \max(N_0(\lambda, m, \ell), 2S_0(f, m, \ell))$. Then for $N := N(\lambda, f, m, \ell)$ it holds that

$$S(\lambda, f, m, \ell, N) = S_0(f, m, \ell) + mN \cdot \mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell)/N < N/2 + N/2 = N \ .$$

Clearly $S_0(f, m, \ell) = O(f)$. Now suppose that $\mathsf{v}(\lambda, N, |\mathsf{ivk}(N)| + \ell) \leq (N^{1-\epsilon}\alpha(\lambda, \ell))$ for all sufficiently large N, λ, ℓ. Let $N'(\lambda, m, \ell) := (2m \cdot \alpha(\lambda, \ell))^{1/\epsilon}$. Then for all m and sufficiently large λ, ℓ, for $N' := N'(\lambda, m, \ell)$,

$$\mathsf{v}(\lambda, N', |\mathsf{ivk}(\lambda, N')| + \ell)/N' < \alpha(\lambda, \ell) \cdot (2m \cdot \alpha(\lambda, \ell))^{-1} = 1/(2m) \ .$$

Hence $N_0 \leq N' = (2m \cdot \alpha(\lambda, \ell))^{1/\epsilon}$. □

The size of the circuit $R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}$ for $N := N(\lambda, f, m, \ell)$ and $k := |\mathsf{ivk}(\lambda, N)| + \ell$ is at most

$$\begin{aligned} S(\lambda, f, m, \ell, N) &= f + O(m\ell) + m \cdot \mathsf{v}(\lambda, N, |\mathsf{ivk}(\lambda, N)| + \ell) \\ &= f + O(N^{1-\epsilon} m\alpha(\lambda, \ell)) \\ &= f + O(f^{1-\epsilon} m \cdot \alpha(\lambda, \ell) + (m \cdot \alpha(\lambda, \ell))^{1/\epsilon}) \ . \end{aligned}$$

In particular if $f = \omega((m \cdot \alpha(\lambda, \ell))^{1/\epsilon})$ then this is $f + o(f)$, and so the stated efficiency bounds hold. □

5.11.5 Security reduction

We establish the security properties in Theorem 5.11.5. We discuss knowledge soundness in Section 5.11.5.1, post-quantum security in Section 5.11.5.2, and zero knowledge in Section 5.11.5.3.

5.11.5.1 Knowledge soundness

In the following, since the extracted transcript T will be a tree, we find it convenient to associate the label $(z^{(u,v)}, \pi^{(u,v)})$ of the unique outgoing edge of a node u with the node u itself, so we refer to this as $(z^{(u)}, \pi^{(u)})$. It is straightforward to transform such a transcript into one that satisfies Definition 5.11.2.

Given a malicious prover $\tilde{\mathbb{P}}$, we will define an extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ that satisfies knowledge soundness. In the process we construct a sequence of extractors $\mathbb{E}_1, \ldots, \mathbb{E}_d$ for $d := d(\Phi)$ (the depth of Φ); $\mathbb{E}_j$ outputs a tree of depth $j + 1$. Let $\mathbb{E}_0(\mathsf{urs})$ run $(\Phi, \mathsf{o}, \pi) \leftarrow \tilde{\mathbb{P}}(\mathsf{urs})$ and output (Φ, T_0), where T_0 is a single node labeled with (o, π). Let $l_{\mathsf{T}}(j)$ denote the vertices of T at depth j; $l_{\mathsf{T}}(0) := \emptyset$ and $l_{\mathsf{T}}(1)$ is the singleton containing the root.

Now we define the extractor $\mathbb{E}_j$ inductively for each $j \in \{1, \ldots, d\}$. Suppose we have already constructed $\mathbb{E}_{j-1}$. We construct a SNARK prover $\tilde{\mathcal{P}}_j$ as follows:

$\tilde{\mathcal{P}}_j(\mathsf{urs})$:

1. Run $(\Phi, \mathsf{T}_{j-1}) \leftarrow \mathbb{E}_{j-1}(\mathsf{urs})$; for each vertex $v \in l_{\mathsf{T}_{j-1}}(j)$, denote its label by $(z^{(v)}, \pi^{(v)})$.
2. Run the indexer $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}})$.
3. Output

$$(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}) := \left(\vec{R}, (\mathsf{ivk}, z^{(v)})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\pi^{(v)})_{v \in l_{\mathsf{T}_{j-1}}(j)}, (\Phi, \mathsf{T}_{j-1})\right)$$

where $\vec{R}$ is the vector $(R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}, \ldots, R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}})$ of the appropriate length.

Next let $\mathcal{E}_{\tilde{\mathcal{P}}_j}$ be the extractor that corresponds to $\tilde{\mathcal{P}}_j$, via the knowledge soundness of the non-interactive argument ARG. Finally the extractor $\mathbb{E}_j$ is defined as follows:

$\mathbb{E}_j(\mathsf{urs})$:

1. Run the extractor $(\vec{\mathbb{i}}, \vec{\mathbb{x}}, \vec{\pi}, \mathsf{aux}, \vec{\mathbb{w}}) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}(\mathsf{urs})$, and parse the auxiliary output aux as (Φ, T').
2. If T' is not a transcript of depth j, abort.
3. Output (Φ, T_j) where T_j is the transcript constructed from T' by doing the following for each vertex $v \in l_{\mathsf{T}'}(j)$:
   - obtain the local data $z^{(v)}_{\mathsf{loc}}$ and input messages $\left(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}}\right)_{i\in[m]}$ from $\mathbb{w}^{(v)}$;
   - append $z^{(v)}_{\mathsf{loc}}$ to the label of v, and if there exists any $z^{(i)}_{\mathsf{in}}$ with $z^{(i)}_{\mathsf{in}} \neq \perp$, attach m children to v where the i-th child is labeled with $(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}})$.

The extractor $\mathbb{E}_{\tilde{\mathbb{P}}}$ runs $(\Phi, \mathsf{T}_d) \leftarrow \mathbb{E}_d(\mathsf{urs})$ and outputs $(\Phi, \mathsf{o}, \pi, \mathsf{T}_d)$, where (o, π) labels the root node.

We now show that $\mathbb{E}_{\tilde{\mathbb{P}}}$ has polynomial size and that it outputs a transcript that is Φ-compliant.

Size of the extractor. $\tilde{\mathcal{P}}_j$ is a circuit of size $|\mathbb{E}_{j-1}| + |\mathcal{I}| + O(2^j)$, so $\mathcal{E}_{\tilde{\mathcal{P}}_j}$ is a circuit of size $e(|\mathbb{E}_{j-1}| + |\mathcal{I}| + O(2^j))$ Then $|\mathbb{E}_j| \leq e(|\mathbb{E}_{j-1}| + |\mathcal{I}| + c \cdot 2^j)$ for some $c \in \mathbb{N}$.

A solution to this recurrence (for $e(n) \geq n$) is $|\mathbb{E}_j| \leq e^{(j)}(|\tilde{\mathbb{P}}| + j \cdot |\mathcal{I}| + 2c \cdot 2^j)$, where $e^{(j)}$ is the function e iterated j times. Hence in particular if $d(\Phi)$ is a constant, $\mathbb{E}_{\tilde{\mathbb{P}}}$ is a circuit of polynomial size.

Correctness of the extractor. Fix a predicate p. We show by induction that, for all $j \in \{0, \ldots, d\}$, the transcript T_j output by $\mathbb{E}_j$ is Φ-compliant up to depth j, that $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(v)}), \pi^{(v)})$ accepts for all $v \in \mathsf{T}$, and that $p(\Phi, \mathsf{o}) = 1$ and $\Phi \in \mathsf{F}$ with probability $\mu - 2j \cdot \kappa(\lambda, m^j)$.

For $j = 0$ the statement is implied by $\mathbb{V}$ accepting, and $p(\Phi, \mathsf{o})$ and $\Phi \in \mathsf{F}$ holding, with probability μ.

Now suppose that $(\Phi, \mathsf{T}_{j-1}) \leftarrow \mathbb{E}_{j-1}$ is such that T_{j-1} is Φ-compliant up to depth $j-1$, that $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(v)}), \pi^{(v)})$ accepts for all $v \in \mathsf{T}_{j-1}$, and that $p(\Phi, \mathsf{o}(\mathsf{T})) = 1$ and $\Phi \in \mathsf{F}$, with probability $\mu - 2(j-1) \cdot \kappa(\lambda, m^{j-1})$. Let $(\vec{\mathbb{i}}, (\mathsf{ivk}_v, z^{(v)})_v, (\pi^{(v)})_v, (\Phi, \mathsf{T}'), \vec{\mathbb{w}}) \leftarrow \mathcal{E}_{\tilde{\mathcal{P}}_j}$.

Let $p'(\mathsf{urs}, \vec{\mathbb{i}}, (\mathsf{ivk}_v, z^{(v)})_v, (\Phi, \mathsf{T}')) = 1$ if:

- $p(\Phi, \mathsf{o}(\mathsf{T}')) = 1$ and $\Phi \in \mathsf{F}$,
- $\mathbb{i}^{(v)} = R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}$ for all v,
- T' is Φ-compliant up to depth $j-1$, and
- for $v \in l_{\mathsf{T}'}(j)$, v is labeled with $(z^{(v)}, \pi^{(v)})$ and $\mathsf{ivk}_v = \mathsf{ivk}$ where $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}})$.

By knowledge soundness, with probability $\mu - 2j \cdot \kappa(\lambda, m^j)$, $p'(\mathsf{urs}, \vec{\mathbb{i}}, (\mathsf{ivk}_v, z_v)_v, (\Phi, \mathsf{T}')) = 1$ and for every vertex $v \in l_{\mathsf{T}'}(j)$, $(R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}, (\mathsf{ivk}_v, z^{(v)}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$. Here we use the predicate and auxiliary output in the knowledge soundness definition of ARG to ensure consistency between the values $z^{(v)}$ and T', and to ensure that T' is Φ-compliant.

Now since $(R^{(\lambda,N,k)}_{\mathcal{V},\Phi,\mathsf{urs}}, (\mathsf{ivk}_v, z^{(v)}), \mathbb{w}^{(v)}) \in \mathcal{R}_{\mathrm{R1CS}}$, we obtain from $\mathbb{w}^{(v)}$ either

- local data $z^{(v)}_{\mathsf{loc}}$ and input messages $\left(z^{(i)}_{\mathsf{in}}, \pi^{(i)}_{\mathsf{in}}\right)_{i\in[m]}$ such that $\Phi(z^{(v)}, z^{(v)}_{\mathsf{loc}}, z^{(1)}_{\mathsf{in}}, \ldots, z^{(m)}_{\mathsf{in}})$ accepts and for all $i \in [m]$ the SNARK verifier $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(i)}_{\mathsf{in}}), \pi^{(i)}_{\mathsf{in}})$ accepts; or
- local data $z^{(v)}_{\mathsf{loc}}$ such that $\Phi(z^{(v)}, z^{(v)}_{\mathsf{loc}}, \perp, \ldots, \perp)$ accepts.

In both cases we append $z^{(v)}_{\mathsf{loc}}$ to the label of v. In the former case we label the children of v with

$(z_{\mathsf{in}}^{(i)}, \pi_{\mathsf{in}}^{(i)})$, and so v is Φ-compliant, and all of its descendants w have that the verifier accepts; i.e. $\mathcal{V}^{(\lambda,N,k)}(\mathsf{urs}, \mathsf{ivk}, (\mathsf{ivk}, z^{(w)}), \pi^{(w)}) = 1$. In the latter case, v has no children and so is Φ-compliant by the base case condition. Hence $\mathsf{T}_j \leftarrow \mathbb{E}_j$ is Φ-compliant up to depth j.

Since $d(\Phi) \leq d$ it must be the case that all vertices v at depth d are in the base case. Hence by induction, $(\Phi, \mathsf{T}) \leftarrow \mathbb{E} = \mathbb{E}_d$ has Φ-compliant T, $p(\Phi, o(\mathsf{T})) = 1$ and $\Phi \in \mathsf{F}$ with probability at least $\mu - 2d \cdot \kappa(\lambda, m^d)$.

5.11.5.2 Post-quantum security

In the quantum setting, $\tilde{\mathbb{P}}$ is taken to be a polynomial-size *quantum* circuit; hence also $\tilde{\mathcal{P}}_j, \mathcal{E}_{\tilde{\mathcal{P}}_j}, \mathbb{E}_j$ are quantum circuits for all j, as is the final extractor $\mathbb{E}$. Our definition of knowledge soundness is such that this proof then generalizes immediately to show security against quantum adversaries. In particular, the only difficulty arising from quantum adversaries is that they can generate their own randomness, whereas in the classical case we can force an adversary to behave deterministically by fixing its randomness. This is accounted for by the distributional requirement placed on the extractor of the argument system ARG.

5.11.5.3 Zero knowledge

The PCD simulator $\mathbb{S}$ operates as follows: define $\mathbb{S}(1^\lambda) := \mathcal{S}(1^\lambda)$; then

$\mathbb{S}(\Phi, z, \tau)$:

1. Compute $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, R_{\mathcal{V},\Phi,\mathsf{urs}}^{(\lambda,N,k)})$ (we assume that τ includes urs).
2. Output $\pi \leftarrow \mathcal{S}(R_{\mathcal{V},\Phi,\mathsf{urs}}^{(\lambda,N,k)}, (\mathsf{ivk}, z), \tau)$.

Zero knowledge then follows immediately from the (adaptive) zero knowledge guarantee of ARG, applied to the honest adversary $\mathcal{A}'$ which runs the honest PCD adversary $(\Phi, z, z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m) \leftarrow \mathcal{A}(\mathsf{urs})$ and outputs $(R_{\mathcal{V},\Phi,\mathsf{urs}}^{(\lambda,N,k)}, (\mathsf{ivk}, z), (z_{\mathsf{loc}}, [z_i, \pi_i]_{i=1}^m))$ where $(\mathsf{ipk}, \mathsf{ivk}) \leftarrow \mathcal{I}(\mathsf{urs}, R_{\mathcal{V},\Phi,\mathsf{urs}}^{(\lambda,N,k)})$. The same argument holds in the quantum setting; in that case, $\mathcal{A}'$ is an honest quantum adversary.

5.12 Implementation of recursive composition

In Section 5.12.1 we describe our implementation work to realize our preprocessing zkSNARK (FRACTAL), and in Section 5.12.2 we describe our implementation work to realize recursive composition.

5.12.1 The preprocessing zkSNARK

Our starting point is `libiop` [132], a library that provides (a) an implementation of the BCS transformation, which compiles any public-coin IOP into a corresponding SNARG by using (instantiated) random oracles; and (b) the *non*-holographic IOPs for R1CS underlying Aurora (Chapter 3) and Ligero [9].

Our work to implement FRACTAL consists of (1) extending the BCS transformation to compile any public-coin *holographic* IOP into a corresponding *preprocessing* SNARG (following Section 5.10); and (2) implementing our efficient holographic IOP for R1CS (following Section 5.8). We discuss each in turn.

(1) From holography to preprocessing. Our transformation from Section 5.10 is a black-box extension of the BCS transformation (see Construction 5.10.3), which made it possible to extend the current implementation of the BCS transformation while re-using much of the existing infrastructure. We modified the generic IOP infrastructure in `libiop` to additionally support expressing *holographic* IOPs, by providing an indexer algorithm (in addition to the prover and verifier algorithms). We modified the transformation to determine if the input IOP is holographic and, if so, to additionally produce an indexer for the argument system, which uses a Merkle tree on the encoded index to produce an index proving key and index verification key. In this case, the prover and verifier for the argument use these keys to produce/authenticate answers about the encoded index, following our construction. Overall, our implementation simultaneously supports the old transformation (from IOP to SNARG) and our new one (from holographic IOP to preprocessing SNARG).

(2) Holographic IOP for R1CS. Our holographic IOP is built from two components (see Theorem 5.8.2): an RS-encoded holographic IOP and a low-degree test. For the latter, we reuse the generic low-degree testing infrastructure in `libiop`: the randomized reduction from testing multiple words to testing single words, and the FRI low-degree test [22]. Our implementation work is about the former component.

Specifically we implement the RS-encoded holographic IOP summarized in Fig. 5.4 (or, more precisely, an optimized and parametrized refinement of it), along with its indexer algorithm (not part of the figure). We reuse the reduction from R1CS to lincheck from the Aurora protocol in `libiop` (as our protocol shares the same reduction). The new key component that we implement is a holographic *multi*-lincheck, which simultaneously supports checking multiple linear relations that were holographically encoded. We believed that this building block of the protocol is of independent interest for the design of holographic proofs.

In addition to enabling sublinear verification, the holographic setting also presents new opportunities for improvements in the concrete efficiency of certain subroutines of the verifier, because we can use the indexer to provide useful precomputed information to the verifier. We leverage such opportunities to precompute various algebraic objects (such as vanishing polynomials), achieving noticeable efficiency improvements.

5.12.2 Designing the verifier's constraint system

In order to recursively compose FRACTAL, we need to design a constraint system that expresses its verifier. We describe a *general* method for designing constraint systems for the verifiers of SNARGs obtained by combining an RS-encoded IOP and the FRI low-degree test (as in Theorem 5.8.2) and then transforming the resulting IOP into a SNARG using Theorem 5.10.1 (henceforth referred to as the "BCS transformation").

The verifier in such SNARGs splits naturally into an "algebraic" part arising from the underlying IOP (hereafter the "IOP verifier") and a "hash-based" part arising from the BCS transforma-

tion (described in Construction 5.10.2, hereafter the "BCS verifier"). We treat them separately: the BCS verifier is discussed in Section 5.12.2.1 and the IOP verifier in Section 5.12.2.2.

5.12.2.1 The BCS verifier

The BCS verifier can be further broken down into two subcomponents. The first is a hashchain that ensures that the IOP verifier's randomness for each round is correctly derived from the Merkle roots (Steps 1 and 2 of the BCS verifier in Construction 5.10.2). The second is the verification of the Merkle tree authentication paths to ensure the validity of the query answers (Step 3 of the BCS verifier in Construction 5.10.2).

The hashchain. The hashchain computation of the BCS verifier is as follows: initialize $\sigma_0 := \mathsf{ivk} \| \mathbb{x}$; then, for each round $i \in \{1, \ldots, \mathsf{k}\}$, derive the randomness $\rho_i := \rho(\sigma_{i-1})$ and use the i-th root ω_i in the argument π to compute $\sigma_i := \rho(\sigma_{i-1} \| \omega_i)$; finally, derive the post-interaction randomness $\rho_{\mathsf{k}+1} := \rho(\sigma_{\mathsf{k}})$. We require a constraint system $\mathcal{S}$ that, given assignments to the variables $(\mathsf{ivk}, \mathbb{x}, \omega_1, \rho_1, \ldots, \omega_{\mathsf{k}}, \rho_{\mathsf{k}})$, is satisfiable if and only if these assignments are consistent with this hashchain computation.

We realize $\mathcal{S}$ via a sponge construction [44], where first ivk and $\mathbb{x}$ are absorbed into the state and then, for each round $i \in \{1, \ldots, \mathsf{k}\}$, the randomness ρ_i is squeezed from the state and the i-th root ω_i is absorbed into the state; the post-interaction randomness $\rho_{\mathsf{k}+1}$ is then squeezed from the state. See Fig. 5.5 for a diagram of this. The size of the constraint system $\mathcal{S}$ is

$$S_{\mathsf{in}}(|\mathsf{ivk}| + |\mathbb{x}|) + \left(\sum_{i=1}^{\mathsf{k}} S_{\mathsf{in}}(|\omega_i|) + S_{\mathsf{out}}(|\rho_i|) \right) + S_{\mathsf{out}}(|\rho_{\mathsf{k}+1}|)$$

where $S_{\mathsf{in}}(n)$ denotes the number of constraints to absorb n field elements and $S_{\mathsf{out}}(n)$ denotes the number of constraints to squeeze n field elements. Naturally, these numbers depend on the particular choice of state transformation that is used to instantiate the sponge (see our evaluation in Section 5.13.2).

The above discussion omits some details. First, in some rounds the prover sends auxiliary information beyond the Merkle root (e.g., the third message of the prover in Fig. 5.4 includes a field element that allegedly equals an evaluation of the polynomial $\hat{t}$), and this auxiliary information must be absorbed together with the Merkle root. Second, Fig. 5.5 suggests that the rate of the sponge is large enough to absorb/squeeze any round's root/randomness with a single application of the state transformation, but this need not be the case, especially if sizes vary across rounds (e.g., we expect $|\mathsf{ivk}| + |\mathbb{x}|$ to be larger than $|\omega_i|$). Indeed, in our implementation we pick the rate of the sponge in such a way as to minimize the overall number of constraints for the hashchain, which means that some information may be absorbed/squeezed across multiple applications of the state transformation.

Authentication paths. For every query made by the IOP verifier to the encoded index or to a proof oracle, the BCS prover provides an authentication path for that query relative to the appropriate Merkle root. Recall that the index verification key ivk contains the root ω_0 of the Merkle tree on the encoded index $\mathbf{I}(\mathbb{i})$; and the argument π contains the roots $\omega_1, \ldots, \omega_{\mathsf{k}}$ of the k Merkle trees that correspond to the k rounds of interaction. For every $i \in \{0, 1, \ldots, \mathsf{k}\}$, we denote by Q_i the queries to the leaves of the i-th Merkle tree, by A_i the claimed query

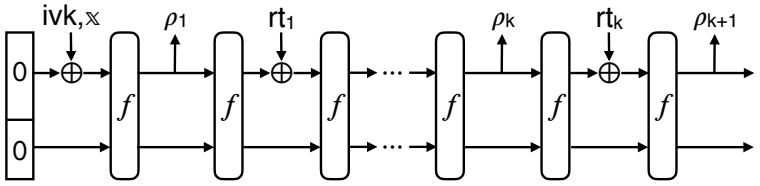


Figure 5.5: We use a sponge construction to realize the hashchain in the BCS verifier.

answers, and by W_i the corresponding auxiliary information to validate them. Both A_i and W_i are provided in the argument π for all i (including $i = 0$). Overall, we require a constraint system $\mathcal{S}$ that, given assignments to the variables $(\omega_i, Q_i, A_i, W_i)_{i=0}^{\mathsf{k}}$, is satisfiable if and only if, for every $i \in \{0, 1, \ldots, \mathsf{k}\}$, the auxiliary information W_i validates the claimed answers in A_i with respect to the queries in Q_i.

Below we describe the basic approach to designing the constraint system. Afterward we describe how to significantly reduce the number of constraints via several optimizations.

The basic approach is to individually validate an authentication path for each query via a separate constraint system. Namely, let ω be a root, $j = \sum_{k=1}^{d} j_k 2^{k-1}$ a query location (in binary representation), a a claimed answer, s a salt used for hiding, and $(u_k)_{k=1}^{d}$ an authentication path. We require a constraint system that, given assignments to the variables $(\omega, j, a, s, (u_k)_{k=1}^{d})$, is satisfiable if and only if the check in the following computation passes: (1) let v_d be the hash of the salted answer $a \| s$; (2) for each $k = d, \ldots, 1$: if the k-th bit of j is 0 then let v_{k-1} be the hash of $v_k \| u_k$, and if instead it is 1 then let v_{k-1} be the hash of $u_k \| v_k$; (3) check that $v_0 = \omega$. See Fig. 5.6 for a diagram of this constraint system.

If we denote by $S_{2\to1}$ the number of constraints to hash two hashes into a single hash, by S_{cswap} the number of constraints for a "controlled swap" on two hashes, and by $S_{\mathsf{leaf}}(n)$ the number of constraints to hash the answer and salt into a single hash, then the number of constraints for the above computation is

$$S_{\mathsf{leaf}}(|a| + |s|) + d \cdot (S_{\mathsf{cswap}} + S_{2\to1}) \ .$$

If we replicate the above strategy for each round $i \in \{0, 1, \ldots, \mathsf{k}\}$ and each query in the query set Q_i, then the total number of constraints to validate all the query answers is:

$$\sum_{i=0}^{\mathsf{k}} |Q_i| \cdot \Big(S_{\mathsf{leaf}}(\alpha_i + \sigma_i) + d_i \cdot (S_{\mathsf{cswap}} + S_{2\to1}) \Big) \ , \tag{5.9}$$

where $\alpha_i \in \mathbb{N}$ denotes the number of field elements to answer a query in round i, $\sigma_i \in \mathbb{N}$ the number of field elements in a salt in round i, and $d_i \in \mathbb{N}$ the depth of the Merkle tree in round i. (Note that σ_0 is always 0 because no hiding is needed for the round that involves the encoded index; σ_i may also be 0 for $i > 0$ in some protocols because zero knowledge may not rely on any query bound to oracles in the i-th round.)

We can do significantly better than Eq. (5.9) if we leverage the structure of query sets, as we now describe.

First, there are known optimizations that increase "leaf size" to reduce argument size [23] that we use to also reduce the number of constraints in our setting. We explain these below.

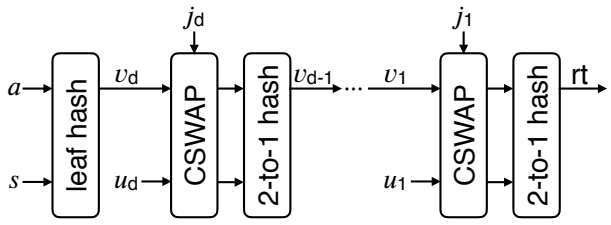


Figure 5.6: Diagram of a constraint system for validating an authentication path.

- *Hash by column.* In protocols derived from RS-encoded IOPs using Theorem 5.8.2, each round's oracles are over the same domain and the IOP verifier queries the same locations across those oracles. This includes the "0-th round oracles", i.e., the oracles in the encoded index $\mathbf{I}(\mathbb{i})$. Hence, for $i = 0, 1, \ldots, \mathsf{k}$, the BCS prover can hash the i-th round oracles *column-wise*: if $\ell_i \in \mathbb{N}$ is the number of oracles in the i-th round then each leaf in the i-th Merkle tree contains a vector in $\mathbb{F}^{\ell_i}$ (the "column") representing one symbol from each of the ℓ_i oracles. Thus a single authentication path suffices to authenticate all the answers from a query to the entire leaf. This not only reduces argument size (fewer authentication paths are included in the argument) but also reduces the number of constraints (fewer authentication paths are validated).
- *Hash by subset.* The low-degree test that we use (see Section 5.12.2.2) yields queries that consider each domain as partitioned into subsets of equal size and each query requests the values of all locations in a subset, i.e., for each round i there is a parameter $m_i \in \mathbb{N}$ for which queries to oracles in the i-th round are always grouped in disjoint subsets of size m_i. Hence the BCS prover can hash all of these locations as part of the same leaf, which now is expanded from a vector in $\mathbb{F}^{\ell_i}$ to a matrix in $\mathbb{F}^{m_i \times \ell_i}$. This reduces the number of authentication paths, and also reduces the depth of the i-th Merkle tree by $\log_2 m_i$ levels.

Second, there are optimizations that pertain only to the goal of reducing the number of constraints, as we now exemplify. Since each oracle is queried at several locations, many authentication paths will overlap in the top layers of the Merkle trees. For argument size, this leads to the optimization of *path pruning* where the argument will contain the minimal collection of hashes that suffices to authenticate a *set* of queries. This optimization (which we continue to use for argument size) does *not* significantly reduce the number of constraints because validating the set of queries still involves re-computing the omitted hashes. Even worse, since query locations are random, we cannot hard-code in the constraint system which hash computations are repeated. We mitigate this problem via the following hybrid approach.

- *Tree cap.* By the pigeonhole principle, any set of authentication paths must overlap towards the top of the tree. To take advantage of this, we modify the Merkle tree in each round i by connecting the vertices at layer t_i (to be chosen later) directly to the root (and discarding the layers in between), so that the root has degree 2^{t_i}. We then compute the Merkle tree root using a "tree cap" hash function taking in 2^{t_i} hashes. Letting $S_{\mathsf{cap}}(n)$ denote the number of constraints for such a hash of n hashes, the total number of constraints across all rounds for the first layer alone is $\sum_{i=0}^{\mathsf{k}} S_{\mathsf{cap}}(2^{t_i})$.
- *Other layers.* For the other layers, we treat the authentication paths as disjoint, and allocate

a separate constraint system to validate the segment of each such path. This amounts to invoking the basic strategy described above, whose cost is summarized in Eq. (5.9), with the modification that the authentication path is reduced from length d_i to length $d_i - t_i$. Note that, in light of the above discussions on answer size, we know that the answer in round i is of size $\alpha_i := m_i \cdot \ell_i$.

The above hybrid approach yields a total number of constraints equal to

$$\sum_{i=0}^{\mathsf{k}} S_{\mathsf{cap}}(2^{t_i}) + |Q_i| \cdot \Big(S_{\mathsf{leaf}}(m_i \cdot \ell_i + \sigma_i) + (d_i - t_i) \cdot (S_{\mathsf{cswap}} + S_{2\to 1}) \Big) \ .$$

The constants t_i are chosen to minimize the above expression. In Section 5.13.2 we discuss the concrete improvements of the hybrid approach over the simplistic approach.

Remark 5.12.1 (arity of the Merkle tree)**.** There are algebraic hash functions for which using Merkle trees with large arity significantly reduces the number of constraints required to check many authentication paths [4, 93]. This comes at the cost of a larger argument size, and our implementation currently does not provide the option of such tradeoffs.

Representing query locations. We have so far assumed that the inputs and outputs of hash functions are *field elements*, as opposed to bits. This is because we instantiate all hash functions via algebraic constructions that require fewer constraints to express (see Section 5.13.2) and also because certain aspects of the verifier are simpler (e.g., the verifier's randomness in the protocol is essentially uniformly random in $\mathbb{F}$). That said, for the *query* part, the verifier does not draw query locations from the whole field $\mathbb{F}$ but, instead, from an evaluation domain contained in $\mathbb{F}$, and we need to obtain a bit-representation of these locations to check Merkle tree authentication paths. Recall also that queries are grouped into subset, and so the location will refer to the subset in the evaluation domain rather than to a single element in the evaluation domain.

We thus perform a bit decomposition of the field elements output by the hash function, split the resulting string into substrings of appropriate size, and regard each substring as a bit-representation of the queried subset. In more detail, for each round $i \in \{0, 1, \ldots, \mathsf{k}\}$, let $L_i \subseteq \mathbb{F}$ be the evaluation domain of round i and recall that queries in round i are on subsets of size m_i. This means that we can obtain $\lfloor \log_2 |\mathbb{F}| \rfloor / (\log_2 |L_i| - \log_2 m_i)$ subsets in L_i for each element in $\mathbb{F}$ output by the hash function. Therefore, if we need to sample q subsets in L_i, the number of field elements that we need to allocate in $\rho_{\mathsf{k}+1}$ is $\lceil \mathsf{q} \cdot \frac{\log_2 |L_i| - \log_2 m_i}{\lfloor \log_2 |\mathbb{F}| \rfloor} \rceil$. Obtaining from these field elements the corresponding bit representations requires about $\mathsf{q} \cdot (\log_2 |L_i| - \log_2 m_i + 2)$ constraints.

We stress that queries across rounds need not be independent. Indeed, for the IOP verifiers that we consider (see Section 5.12.2.2), it will hold that each round receives the same number of queries (informally, there exists q such that $\mathsf{q} = |Q_0| = |Q_1| = \cdots = |Q_{\mathsf{k}}|$) and all the queries are correlated in that the q subsets for each round can all be derived from q samples of a certain length.

5.12.2.2 The IOP verifier

We describe the design of a constraint system that can express the verifier of any (holographic) IOP derived from an RS-encoded (holographic) IOP and a low-degree test, according to the construction underlying Theorem 5.8.2. Informally, the RS-encoded (holographic) IOP is an interactive reduction that leads to a set of algebraic claims about the prover's oracles (and possibly also about the encoded index); and the low-degree test is an interactive protocol that is used to ensure that these algebraic claims hold.

Outline. Let $(\mathbf{I}, \mathbf{P}, \mathbf{V})$ be a holographic IOP for an indexed relation $\mathcal{R}$ constructed via Theorem 5.8.2. Note that $\mathcal{R}$ need not be the R1CS indexed relation. Below we recall the two ingredients of the construction.

- A $\mathsf{k}^{\mathcal{R}}$-round RS-encoded holographic IOP $(\mathbf{I}_{\mathcal{R}}, P_{\mathcal{R}}, V_{\mathcal{R}}, \{\vec{d}_{\mathbf{I}}, \vec{d}_{\mathbf{P},1}, \ldots, \vec{d}_{\mathbf{P},\mathsf{k}^{\mathcal{R}}}\})$ over a domain L, with maximum degree $(d_{\mathsf{c}}, d_{\mathsf{e}})$, for the indexed relation $\mathcal{R}$. In each of $\mathsf{k}^{\mathcal{R}}$ rounds, the RS-hIOP verifier $V_{\mathcal{R}}$ sends randomness and the RS-hIOP prover $P_{\mathcal{R}}$ sends an oracle; after the interaction, the RS-hIOP verifier $V_{\mathcal{R}}$ outputs a set of rational constraints (the algebraic claims, see Definition 5.4.1). We view the RS-hIOP verifier $V_{\mathcal{R}}$ as a function that maps an instance $\mathbb{x}$ and randomness $\boldsymbol{\rho}^{\mathcal{R}}$ to a set of rational constrains $\mathfrak{C}$.

- A $\mathsf{k}^{\mathrm{LDT}}$-round low-degree test $(\mathbf{P}_{\mathrm{LDT}}, \mathbf{V}_{\mathrm{LDT}})$ for the Reed–Solomon code $\mathrm{RS}\,[L, d_{\mathsf{c}}]$. In each of $\mathsf{k}^{\mathrm{LDT}}$ rounds, the LDT verifier $\mathbf{V}_{\mathrm{LDT}}$ sends randomness and the LDT prover $\mathbf{P}_{\mathrm{LDT}}$ sends an oracle; after the interaction, the LDT verifier $\mathbf{V}_{\mathrm{LDT}}$ makes $\mathsf{q}^{\mathrm{LDT}}$ queries, and then accepts or rejects. We can view the LDT verifier $\mathbf{V}_{\mathrm{LDT}}$ as two algorithms: a *query algorithm* $\mathbf{V}_{\mathrm{LDT}}.\mathrm{Q}$ such that $(Q_0, Q_1, \ldots, Q_{\mathsf{k}^{\mathrm{LDT}}}) := \mathbf{V}_{\mathrm{LDT}}.\mathrm{Q}(\boldsymbol{\rho}^{\mathrm{LDT}})$ are the queries to the tested oracle and the $\mathsf{k}^{\mathrm{LDT}}$ prover oracles on randomness $\boldsymbol{\rho}^{\mathrm{LDT}}$; and a *decision algorithm* $\mathbf{V}_{\mathrm{LDT}}.\mathrm{D}$ such that $\mathbf{V}_{\mathrm{LDT}}.\mathrm{D}(A_0, A_1, \ldots, A_{\mathsf{k}^{\mathrm{LDT}}}, \boldsymbol{\rho}^{\mathrm{LDT}})$ is the decision of the verifier given answers to the queries and the same randomness.

We need to design a constraint system to express the computation of the holographic IOP verifier $\mathbf{V}$, and so we are faced with three sub-tasks: (1) design a constraint system for the RS-hIOP verifier $V_{\mathcal{R}}$; (2) design a constraint system for the LDT verifier $\mathbf{V}_{\mathrm{LDT}}$; (3) combine these two into a constraint system for $\mathbf{V}$.

We review features of the construction in Theorem 5.8.2 relevant for designing the constraint system.

- *Randomness.* The verifier $\mathbf{V}$ has $\mathsf{k}^{\mathcal{R}} + \mathsf{k}^{\mathrm{LDT}}$ rounds of interaction where the first $\mathsf{k}^{\mathcal{R}}$ rounds are for $V_{\mathcal{R}}$ and the remaining $\mathsf{k}^{\mathrm{LDT}}$ rounds are for $\mathbf{V}_{\mathrm{LDT}}$. This means that we can split the randomness $\boldsymbol{\rho}$ of $\mathbf{V}$ into randomness $\boldsymbol{\rho}^{\mathcal{R}}$ for $V_{\mathcal{R}}$ and randomness $\boldsymbol{\rho}^{\mathrm{LDT}}$ for V_{LDT}.

- *Domains.* The oracles in the encoded index and in the first $\mathsf{k}^{\mathcal{R}}$ rounds are all over the domain L, while oracles in the other $\mathsf{k}^{\mathrm{LDT}}$ rounds are over domains determined by the LDT.

- *Queries.* The queries to the oracles in the encoded index and in the first $\mathsf{k}^{\mathcal{R}}$ rounds are all the same, i.e., they are specified by the query set $Q_0 \subseteq L$ for the tested oracle determined by the LDT verifier.

- *Tested oracle.* The low-degree test is invoked on a "virtual oracle" $f\colon L \to \mathbb{F}$ defined as a random linear combination of rational constraints output by the RS-hIOP verifier. Namely, if $((N_k, D_k, d_k))_{k=1}^r$ are the rational constraints output by $V_{\mathcal{R}}(\mathbb{x}; \boldsymbol{\rho}^{\mathcal{R}})$, $\alpha_1, \ldots, \alpha_k$ are the random coefficients, and $f_1, \ldots, f_\ell\colon L \to \mathbb{F}$ are the oracles sent by the RS-hIOP prover across the $\mathsf{k}^{\mathcal{R}}$ rounds, then f is defined as follows:

$$\forall\, a\,,\ f(a) := \sum_{k=1} \alpha_k \cdot \frac{N_k(a, f_1(a), \ldots, f_\ell(a))}{D_k(a)}\ .$$

  The low-degree test will read f at the query set Q_0, which means that all oracles $f_1, \ldots, f_\ell$ will also be read at Q_0, and their answers must be combined according to the rational constraints and random coefficients.

(1) RS-hIOP. First we note that the structure of the interactive phase of the RS-hIOP for $\mathcal{R}$ determines what the hashchain described in Section 5.12.2.1 needs to squeeze and absorb for the first $\mathsf{k}^{\mathcal{R}}$ rounds. In the case of our RS-hIOP for R1CS this round information can be directly read off from Fig. 5.4.

We now turn to discussing the constraint system associated to $V_{\mathcal{R}}$, which is tasked to evaluate the rational constraints output by $V_{\mathcal{R}}$ at a set of locations $Q \subseteq L$ (the queries for the oracle tested by the LDT).

Suppose that the number of oracles sent by the RS-hIOP prover across the $\mathsf{k}^{\mathcal{R}}$ rounds is ℓ, and suppose that the number of rational constraints output by the RS-hIOP verifier is r. We seek a constraint system that, given as input an instance $\mathbb{x}$, randomness $\boldsymbol{\rho}^{\mathcal{R}}$, query set Q, answers from all oracles $(\beta_{a,j})_{a\in Q, j\in[\ell]}$ and claimed evaluations $(\gamma_{a,k})_{a\in Q, k\in[r]}$, is satisfiable if and only if, letting $((N_k, D_k, d_k))_{k=1}^r$ be the rational constraints output by $V_{\mathcal{R}}(\mathbb{x}; \boldsymbol{\rho}^{\mathcal{R}})$, it holds that

$$\forall\, a \in Q\ ,\ \forall\, k \in [r]\ ,\ \gamma_{a,k} = \frac{N_k(a, \beta_{a,1}, \ldots, \beta_{a,\ell})}{D_k(a)}\ .$$

In all known RS-encoded protocols, including the RS-hIOP for R1CS in Fig. 5.4, the rational constraints output by the RS-hIOP verifier depend on the instance $\mathbb{x}$ and randomness $\boldsymbol{\rho}^{\mathcal{R}}$ in an algebraic way, in the sense that we can view $\mathbb{x}$ and $\boldsymbol{\rho}^{\mathcal{R}}$ as auxiliary variables of the arithmetic circuits $(N_k)_{k=1}^r$. This means that the cost to check all the equations above is

$$|Q| \cdot \left(\sum_{k=1}^{r} |N_k| + |D_k| + 1 \right)\ ,$$

where $|N_k|$ and $|D_k|$ denote the sizes of the arithmetic circuits for N_k and D_k. (The additive 1 accounts for checking the equality given variables that contain the outputs of the two arithmetic circuits.) Note that these complexity measures are related to, but different from, the query evaluation time defined in Section 5.4.1: query evaluation time is a *uniform* complexity measure, whereas circuit size is *non-uniform*.

In our implementation we additionally reuse sub-computations across constraint systems and across evaluation points to reduce the size of the constraint system. For example, if $D_k = D_{k'}$ for distinct k, k' then we know that we only need to compute $D_k(a)$ once; similarly if N_k and

$N_{k'}$ share sub-computations. One can verify that there are several such opportunities for the RS-hIOP for R1CS in Fig. 5.4.

(2) Low-degree test. The low-degree test that we use in this paper is FRI [22], which is a logarithmic-round logarithmic-query protocol. Below we describe a constraint system that represents the FRI verifier.

Let L be the domain of the oracle to be tested (i.e., the domain of the RS-hIOP). The size of L induces a list of "localization parameters" $(\eta_1, \ldots, \eta_{\mathsf{k}^{\mathrm{LDT}}})$ which in turn induces a list of domains $(L_1, \ldots, L_{\mathsf{k}^{\mathrm{LDT}}})$ with progressively smaller sizes, $|L_i| = |L_{i-1}|/2^{\eta_i} = |L|/2^{\eta_1+\cdots+\eta_i}$ with $L_0 := L$. Each domain L_i is obtained from L_{i-1} as the image of a 2^{η_i}-to-1 map h_i that maps cosets of size 2^{η_i} in L_{i-1} to single points in L_i. Any coset U_0 of size 2^{η_1} in the domain $L_0 = L$ determines $\mathsf{k}^{\mathrm{LDT}} - 1$ cosets $(U_1, \ldots, U_{\mathsf{k}^{\mathrm{LDT}}-1})$ of respective sizes $(2^{\eta_2}, \ldots, 2^{\mathsf{k}^{\mathrm{LDT}}})$ contained in $(L_1, \ldots, L_{\mathsf{k}^{\mathrm{LDT}}-1})$ as follows: for each $i \in \{1, \ldots, \mathsf{k}^{\mathrm{LDT}} - 1\}$, U_i is the unique coset of size $2^{\eta_{i+1}}$ that contains the point $h_i(U_{i-1})$.

We separately address the interactive phase and the query phase.

- *Interactive phase.* For $i \in \{1, \ldots, \mathsf{k}^{\mathrm{LDT}}\}$, in round i the FRI verifier sends a random field element α_i and the FRI prover replies with an oracle $f_i \colon L_i \to \mathbb{F}$ if $i < \mathsf{k}^{\mathrm{LDT}}$, or with a (non-oracle) message containing the coefficients of a polynomial $\hat{f}_{\mathsf{k}^{\mathrm{LDT}}}(X)$ if $i = \mathsf{k}^{\mathrm{LDT}}$. If the degree to be tested is d then the degree of $\hat{f}_{\mathsf{k}^{\mathrm{LDT}}}(X)$ is $d_{\mathsf{k}^{\mathrm{LDT}}} := d/2^{\eta_1+\cdots+\eta_{\mathsf{k}^{\mathrm{LDT}}}}$. Queries to domain L_{i-1} are grouped in cosets of size 2^{η_i}. Hence larger localization parameters lead to fewer rounds, at the expense of querying larger cosets. (Choosing these parameters well is crucial to minimizing constraint complexity, as we discuss in Section 5.13.2.)

  Since the FRI verifier is public-coin, its interactive phase does not yield any special constraints. However the specifics of the interaction affect the hashchain described in Section 5.12.2.1, which is responsible to squeeze verifier randomness and absorb prover messages. We deduce that, for each round $i \in \{1, \ldots, \mathsf{k}^{\mathrm{LDT}} - 1\}$: the hashchain is required to squeeze a single field element and then to absorb a single Merkle root, and the depth of the corresponding Merkle tree is $\log_2 |L| - (\eta_1 + \cdots + \eta_{i+1})$. In the last round ($i = \mathsf{k}^{\mathrm{LDT}}$), the hashchain is required to squeeze a single field element and then to absorb $d_{\mathsf{k}^{\mathrm{LDT}}} + 1$ field elements.

- *Query phase.* The FRI verifier repeats the following q times, for a number q that controls soundness error.

  - *Queries.* The FRI verifier samples a random coset U_0 of size 2^{η_1} in the domain $L_0 = L$, and reads the values of the oracle to be tested at U_0. The coset U_0 determines, for each $i \in \{1, \ldots, \mathsf{k}^{\mathrm{LDT}} - 1\}$, a coset U_i of size $2^{\eta_{i+1}}$ in the domain L_i, and the FRI verifier reads the values of the oracle f_i in round i at U_i. Finally, the FRI verifier also reads all the $d_{\mathsf{k}^{\mathrm{LDT}}} + 1$ coefficients of the polynomial sent in round $\mathsf{k}^{\mathrm{LDT}}$.
  - *Decision.* For each $i \in \{0, 1, \ldots, \mathsf{k}^{\mathrm{LDT}} - 1\}$, let $p_i(X)$ be the polynomial of degree less than $2^{\eta_{i+1}}$ that equals the interpolation of the values read for the i-th coset U_i. Let $\hat{f}_{\mathsf{k}^{\mathrm{LDT}}}(X)$ be the polynomial of degree $d_{\mathsf{k}^{\mathrm{LDT}}}$ sent by the prover in the last round (round $i = \mathsf{k}^{\mathrm{LDT}}$).

    The FRI verifier performs the following $\mathsf{k}^{\mathrm{LDT}}$ consistency checks: for each $i \in \{1, \ldots, \mathsf{k}^{\mathrm{LDT}} - 1\}$, check that $p_{i-1}(\alpha_i) = f_i(h_i(U_{i-1}))$; also check that $p_{\mathsf{k}^{\mathrm{LDT}}-1}(\alpha_{\mathsf{k}^{\mathrm{LDT}}}) = \hat{f}_{\mathsf{k}^{\mathrm{LDT}}}(\alpha_{\mathsf{k}^{\mathrm{LDT}}})$.

The implication of the first item above to the constraint system is that the hashchain described in Section 5.12.2.1 needs to squeeze enough field elements to determine q samples of starting cosets in the domain $L_0 = L$ (with each sample indexed in binary as already discussed). Moreover, the constraint system needs to check, for each starting coset U_0, that the remaining $\mathsf{k}^{\mathrm{LDT}} - 1$ cosets U_i are correctly chosen. For this, since h_i has degree 2^{η_i}, we need at most 2^{η_i} constraints. Hence, the total constraint cost for checking q lists of cosets is $\mathsf{q} \cdot \sum_{i=1}^{\mathsf{k}^{\mathrm{LDT}}} 2^{\eta_i}$. (In fact, we can avoid this cost altogether: by choosing an appropriate bit representation, we can obtain the bit decomposition of the index of coset U_i by truncating the bit decomposition of the index of coset U_{i-1}, in which case *no* constraints are needed.)

The implication of the second item above to the constraint system is that the FRI verifier, for each of q runs, needs to evaluate the interpolation of $\mathsf{k}^{\mathrm{LDT}}$ cosets at a single point and also evaluate the polynomial contained in the last message at a single point. This number of constraints for this is

$$\mathsf{q} \cdot \left(S_{\mathsf{eval}}\left(\frac{d}{2^{\eta_1+\cdots+\eta_{\mathsf{k}^{\mathrm{LDT}}}}}\right) + \sum_{i=1}^{\mathsf{k}^{\mathrm{LDT}}} S_{\mathsf{lde}}(2^{\eta_i}) \right) \quad ,$$

where $S_{\mathsf{eval}}(n) = n$ is the number of constraints to evaluate a polynomial of degree n (say, via a constraint system that follows Horner's method), and $S_{\mathsf{lde}}(n) = 2n + O(\log n)$ is the number of constraints to evaluate at a single point the interpolation of a function defined over a size-n coset. We justify this latter cost below, because the design of the constraint system for interpolation requires some care.

Coset interpolation. We require a constraint system that, given the identifier of a coset S in a domain L, a function $f\colon S \to \mathbb{F}$, an evaluation point $\gamma \in \mathbb{F}$, and a claimed evaluation $v \in \mathbb{F}$, is satisfiable if and only if $v = \sum_{a \in S} f(a) L_{a,S}(\gamma)$, where $\{L_{a,S}(X)\}_{a \in S}$ are the Lagrange polynomials for S.

We describe a constraint system of size $2|S| + O(\log(|S|))$. Given the Lagrange coefficients, we can compute the inner product of the function and the Lagrange coefficients with $|S|$ constraints. This leaves $|S| + O(\log(|S|))$ constraints to compute the Lagrange coefficients, as we discuss below.

A simplistic approach would be to deduce the coefficients of each Lagrange polynomial $\{L_{a,S}(X)\}_{a \in S}$, hardcode these coefficients in the constraint system, and then let the constraint system compute $\{L_{a,S}(\gamma)\}_{a \in S}$ for the given evaluation point $\gamma \in \mathbb{F}$. However, the choice of coset S is *not* known at "compile time" (when constructing the constraint system) because the identifier of S in the domain L is an input to the constraint system. We now explain how to efficiently compute all the evaluations without "generically" deriving the coefficients of each Lagrange polynomial (which would be much more expensive).

Observe that, at compile time, we know *some* information about S: the *base coset* (i.e., subgroup) S^* from which the coset S is derived as a shift (S^* need not be in L). Namely, in the additive case $S = S^* + \xi$ for some $\xi \in \mathbb{F}$, and in the multiplicative case $S = \xi S^*$ for some $\xi \in \mathbb{F}$. Thus the identifier of S in L can be viewed as encoding the shift ξ that determines S from S^*. This is useful because: (a) the vanishing polynomial of a coset S is closely related to the

vanishing polynomial of its base coset S^*; and (b) each Lagrange polynomial can be expressed via the vanishing polynomial $v_S(X)$ and its derivative $v'_S(X)$. Specifically, for every $a \in S$, $L_{a,S}(X) = \frac{1}{v'_S(a)} \cdot \frac{v_S(X)}{X-a}$. This enables us to hardcode in the constraint system information about the base coset S^*, and task the constraint system with a cheap computation that depends on the shift ξ.

We describe this approach for the additive and multiplicative cases separately.

- **Additive case.** The derivative $v'_S(X)$ is a constant $c_{S^*} \in \mathbb{F}$ that only depends on the base coset S^*. Hence all the values $\{v'_S(a)\}_{a \in S}$ (and their inverses) are known at compile time, as they all equal c_{S^*}. The polynomial $v_S(X)$ equals $v_{S^*}(X - \xi)$, which has $O(\log(|S|))$ non-zero monomials. Hence, if we hardcode the polynomial v_{S^*} in the constraint system, we can compute $v_S(\gamma) = v_{S^*}(\gamma - \xi) \in \mathbb{F}$, and also $v_S(\gamma)/c_{S^*} = v_{S^*}(\gamma - \xi)/c_{S^*} \in \mathbb{F}$ (common to all Lagrange coefficients) with $O(\log(|S|))$ constraints.

  Next, note that each element $a \in S$ can be written as $a = a^* + \xi$ for a corresponding element $a^* \in S^*$. This means that $\{X - a\}_{a \in S} = \{X - a^* - \xi\}_{a^* \in S^*}$, where the elements a^* are hardcoded in the constraint system and ξ is an input to the constraint system. In particular, given $v_{S^*}(\gamma - \xi)/c_{S^*} \in \mathbb{F}$ we can compute $\{L_{a,S}(\gamma)\}_{a \in S} = \{\frac{v_{S^*}(\gamma-\xi)}{c_{S^*}} \cdot \frac{1}{\gamma - a^* - \xi}\}_{a^* \in S^*}$ with $|S|$ additional constraints.

- **Multiplicative case.** The polynomial $v_S(X)$ is the polynomial $X^{|S|} - \xi^{|S|}$, and its derivative $v'_S(X)$ is the polynomial $|S|X^{|S|-1}$; recall that $|S| = |S^*|$ and so this quantity is known at compile time. Moreover, each element $a \in S$ can be written as $a = \xi a^*$ for a corresponding element $a^* \in S^*$. Therefore we can re-write each Lagrange polynomial as:

$$\begin{aligned} L_{a,S}(X) &= \frac{1}{v'_S(a)} \cdot \frac{v_S(X)}{X - a} \\ &= \frac{1}{|S|(\xi a^*)^{|S|-1}} \cdot \frac{X^{|S|} - \xi^{|S|}}{X - \xi a^*} \\ &= \frac{1}{|S|(\xi a^*)^{|S|}} \cdot \frac{X^{|S|} - \xi^{|S|}}{X(\xi a^*)^{-1} - 1} \\ &= \frac{1}{|S|\xi^{|S|}} \cdot \frac{X^{|S|} - \xi^{|S|}}{X(\xi a^*)^{-1} - 1} \ . \end{aligned}$$

  The above expression leads to the following strategy. The constraint system first uses the shift ξ and evaluation point γ to compute, via $O(\log(|S|))$ constraints, the value $\frac{\gamma^{|S|} - \xi^{|S|}}{|S|\xi^{|S|}}$; and also one constraint to compute $\gamma\xi^{-1}$. Then, the constraint system computes the values $\{L_{a,S}(\gamma)\}_{a \in S} = \{\frac{\gamma^{|S|} - \xi^{|S|}}{|S|\xi^{|S|}} \cdot \frac{1}{\gamma(\xi a^*)^{-1} - 1}\}_{a^* \in S^*}$ with $|S|$ additional constraints.

5.13 Evaluation

In Section 5.13.1 we evaluate our implementation of the preprocessing zkSNARK, and in Section 5.13.2 we evaluate our implementation of recursive composition.

All reported measurements were run in single-threaded mode on a machine with an Intel Xeon 6136 CPU at $3.0\,\mathrm{GHz}$ with $252\,\mathrm{GB}$ of RAM (no more than $32\,\mathrm{GB}$ of RAM were used in any experiment).

5.13.1 Performance of the preprocessing zkSNARK

We report on the performance of FRACTAL, the preprocessing zkSNARK for R1CS that we have implemented by extending `libiop` as described in Section 5.12.1. We configure our implementation to achieve 128 bits of security, for constraints expressed over a prime field of 181 bits. This field choice is illustrative, as the only requirement on the field is that it should contain suitable subgroups for us to use.

In Fig. 5.7 we report the costs for several efficiency measures, and for each measure also indicate how much of the cost is due to the probabilistic proof and how much is due to the cryptographic compiler. The costs depend on the number of constraints m in the R1CS instance,[14] and so we report how the costs change as we vary m over the range $\{2^{10}, 2^{11}, \ldots, 2^{20}\}$. Below, by *native execution time* we mean the time that it takes to check that an assignment satisfies the constraint system, and by *native witness size* we mean the number of bytes required to represent an assignment to the constraint system.

- *Indexer time.* In the upper left, we plot the running time of the indexer, as absolute cost (top graph) and as relative cost when compared to native execution time (bottom graph). Indexer times range from fractions of a second to several minutes, and the plot confirms the quasilinear complexity of the indexer. Indexer time is dominated by the cost of running the underlying HIOP indexer.

- *Prover time.* In the upper right, we plot the running time of the prover, as absolute cost (top graph) and as relative cost when compared to native execution time (bottom graph). Prover times range from fractions of a second to several minutes, and the plot confirms the quasilinear complexity of the prover. Prover time is dominated by the cost of running the underlying HIOP prover.

- *Argument size.* In the lower left, we plot argument size, as absolute cost (top graph) and as relative cost when compared to native witness size (bottom graph). Argument sizes range from $80\,\mathrm{kB}$ to $160\,\mathrm{kB}$ with compression (argument size is smaller than native witness size) occurring for $m \geq 4,000$, and the plot confirms the polylogarithmic complexity of the argument. Argument size is dominated by the cryptographic digests to authenticate query answers.

[14] More precisely, the costs in general depend on (a) m, the number of constraints (i.e., number of rows in each matrix); (b) n, the number of variables (i.e., number of columns in each matrix); (c) m, the number of non-zero entries in a matrix; and (d) k, the number of public inputs. The number of constraints m and the number of variables n are typically approximately equal, and indeed in this paper we have assumed for simplicity that $m = n$ (the matrices in Definition 5.3.2 are square); so we only keep track of m. The number of non-zero entries m is typically within a small factor of m, and in our experiments m/m is approximately 1. Finally, the number of public inputs k is at most n, and in typical applications it is much smaller than n, so we do not focus on it.

- *Verifier time.* In the lower right, we plot the running time of the verifier, as absolute cost (top graph) and as relative cost when compared to native execution time (bottom graph). Verifier times are several milliseconds and become faster than native execution for $m \geq 65,000$, and the plot confirms the polylogarithmic complexity of the verifier. Verifier time is dominated by the cost of running the underlying HIOP verifier.

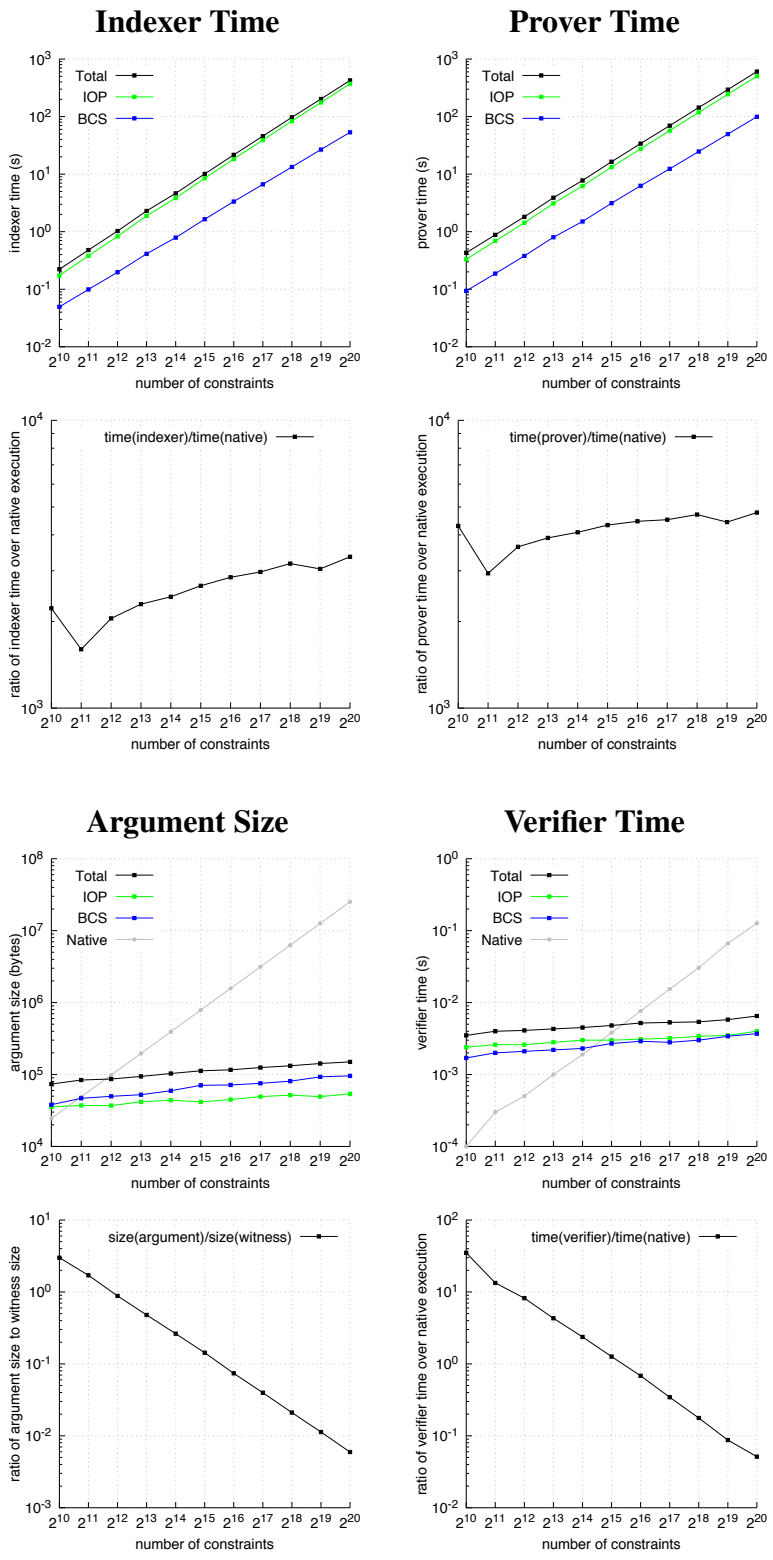


Figure 5.7: Performance of FRACTAL.

Finally, in Fig. 5.8, we compare FRACTAL with the state of the art in several types of zkSNARKs for R1CS:

(1) *Aurora*, a non-preprocessing zkSNARK in the (quantum) random oracle model (Chapter 3);
(2) *Groth16*, a preprocessing zkSNARK with circuit-specific SRS [95];
(3) *Marlin*, a preprocessing zkSNARK with universal SRS [65].

The first protocol is configured the same as our protocol (128 bits of security over a prime field of 181 bits), and the implementation that we use is from `libiop` [132]. The second and third protocols require a choice of pairing-friendly elliptic curve, which we take to be `bls12-381`; the implementation of the second protocol is from `libzexe` [133] and the implementation of the third protocol is from `marlin` [134].

While informative, the comparison should be considered qualitative, because the protocols expose R1CS defined over different prime fields, which means that the same statement may require a different number of constraints when expressed over one field versus another.

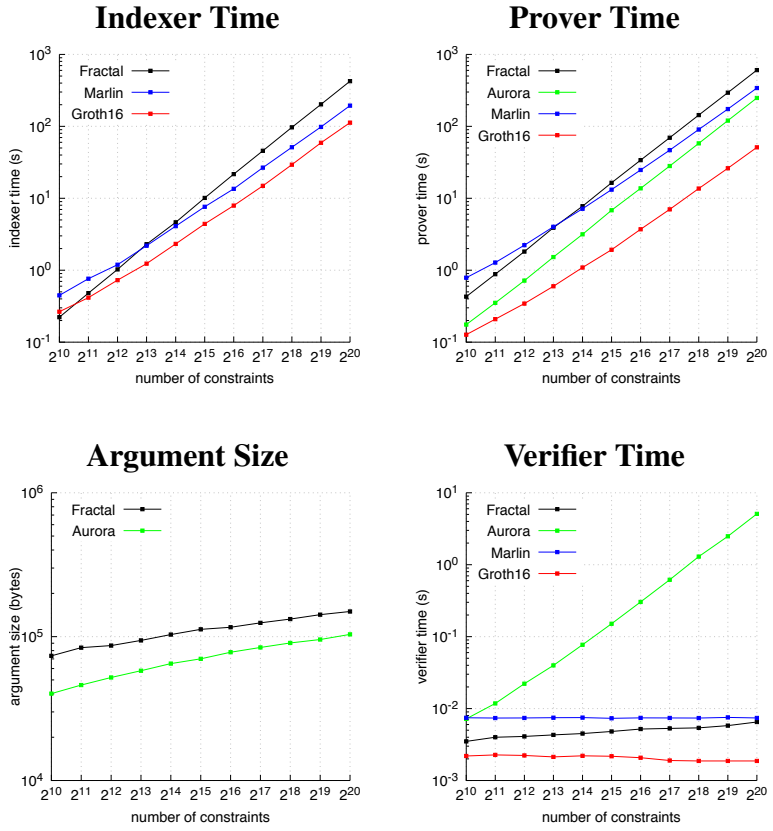


Figure 5.8: Comparison across several zkSNARKs for R1CS. The argument size for [95] is 192 B and for [65, Marlin] is 880 B; they are not plotted in the argument size graph because they are *much* smaller than the argument sizes for the other protocols (which differ in that they are post-quantum and transparent). Note that the setup algorithm for [95] is plotted in the indexer graph because it also serves as an indexer.

5.13.2 Performance of recursive composition

We report on the performance of recursive composition based on FRACTAL. Recall from Section 5.11 that the quantity governing the efficiency of recursion is the complexity of the verifier when expressed as an R1CS constraint system. For R1CS, the we measure complexity by the *number of constraints* and the *total nonzero coefficients* (which are sometimes in contention); we refer to these quantities together as the *verifier size*.

If we were to directly translate the "native" verifier that we evaluate in the previous section into a constraint system, it would be much too large to prove. The primary culprit for this is the choice of hash function. In the previous section we chose a hash function with fast native execution; unfortunately, it is large when represented as an R1CS constraint system. For recursive composition, we will choose a hash function that has a much smaller R1CS representation. Since this hash function is much slower to execute natively, this results in an increase in the proving time per constraint; however, this increase is modest compared to the significant saving arising from the smaller constraint system.

In the remainder of this section we discuss our choice of hash function and how we set its parameters; we then discuss an optimization that significantly improves the native execution time of the hash function without increasing the constraint system size.

Choice of hash function. We instantiate the various hash functions required in Section 5.12.2.1 via different parameterizations of *Poseidon* [93]. This is a sponge hash function [44], which means that it maintains a state that is split into two parts: the *rate* part of the state, which is used to absorb inputs and squeeze outputs; and the *capacity* part of the state, the size of which determines the security of the sponge.

We will set the Poseidon parameters for each hash function in order to jointly minimize the number of constraints and the number of nonzero coefficients. We assume our field is such that $\lfloor \log(|\mathbb{F}|) \rfloor \geq 2\lambda$, which allows us to set the capacity to be one field element, and the hash output size as one field element. It remains to choose the rate, and the Poseidon parameter α, which controls the degree of the S-Box permutations.

We set $\alpha = 17$ for all of our hash functions. This is higher than typical instantiations of Poseidon. This choice reduces the number of rounds of the hash function, which greatly reduces the number of non-zero R1CS coefficients in exchange for a modest increase in the number of R1CS constraints. We find empirically that the number of non-zero R1CS coefficients per element of rate is minimized for $\alpha = 17$ when the rate is 10. Correspondingly, we set the rate of each hash function to be the minimum of 10 and the number of elements that must be absorbed/squeezed in a single execution of the hash function.

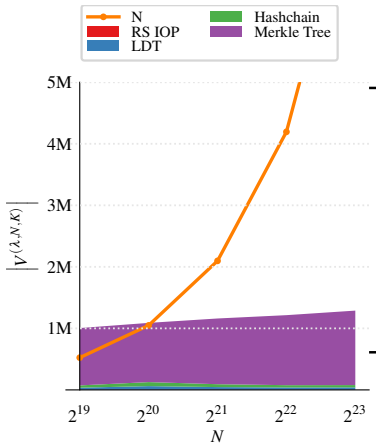
Prover execution time. The recursive prover's running time is affected by the native execution time of the hash function. The direct implementation of Poseidon as our hash function is too slow, causing a $100\times$ slowdown to the recursive prover. This is due to every round requiring the multiplication of a vector by a random MDS matrix. To address this, we instead rely on MDS matrices with 'light-weight circuits' [76, 79], i.e., MDS matrices with small entries (when viewed as integer matrices), for which matrix-vector products can be computed without field multiplications. This leads to a $10\times$ performance improvement in hash execution time, which reduces the slowdown versus the 'standalone' prover to $10\times$.

Verifier size. Recall from Definition 5.11.4 that $\mathcal{V}^{(\lambda,N,k)}$ denotes an R1CS instance expressing

the computation of the SNARK verifier $\mathcal{V}$, for security parameter λ, when checking R1CS instances with at most N constraints and an explicit input of size at most k. Our goal is to minimize the size of $\mathcal{V}^{(\lambda,N,k)}$. For our evaluation we fix the security parameter $\lambda := 128$ and the instance size $k := 100$, and measure how $|\mathcal{V}^{(\lambda,N,k)}|$ varies with N.

Given our choice of hash function parameters, the number of non-zero R1CS coefficients is always within a factor of two of the number of R1CS constraints. Hence for simplicity, below we report only the number of R1CS constraints. This suffices for finding the "recursion threshold", the smallest value N^* for which $\mathcal{V}^{(\lambda,N^*,k)}$ has fewer than N^* constraints (and hence the smallest N^* that admits recursion).

In Section 5.12.2 we described our design of a generic verifier circuit, which left several parameters unspecified (e.g., the number of commitments sent by the prover in a particular round, the number of field elements sent by the verifier in a particular round, the specific rational constraints, and so on). In our implementation we specialize this design to the verifier for FRACTAL to obtain a constraint system that expresses its correct execution. In Fig. 5.9 we plot the measured size of $\mathcal{V}^{(\lambda,N,k)}$ against the number of constraints N it is checking. The graph shows that the recursion threshold is at most 2 million: for all N greater than 2 million, $|\mathcal{V}^{(\lambda,N,k)}| \leq N$. Since we are able to prove constraint systems of this size, this demonstrates feasibility of recursion in our implementation. We are optimistic that further optimizations will further reduce the size of the verifier, and hence also the recursion threshold.



N	2^{20}	2^{21}	2^{22}	2^{23}
RS-IOP	2,948	3,036	3,124	3,212
LDT	58,586	42,658	36,190	37,598
Hashchain	65,184	46,878	37,566	37,566
Merkle Tree	962,544	1,068,122	1,137,818	1,210,418
- leaf hash	235,840	235,642	284,746	309,298
- 2-to-1 hash	663,830	760,624	781,216	829,264
- cap hash	62,874	71,856	71,856	71,856
Total	**1,132,666**	**1,210,292**	**1,264,299**	**1,338,406**

Figure 5.9: On the left we plot, in a linear-log scale, the number of constraints to express the FRACTAL verifier as a function of the number of constraints it is checking (N), using areas of different colors to denote contributions from different constraint types. In the same graph we also plot the number of checked constraints (the function $N \mapsto N$), which shows how the number of constraints for the verifier (which is $\text{polylog}_{\lambda,k}(N)$) grows much slower than N, giving a cross-over point. On the right, we provide several data points for different values of N.

Bibliography

[1] The Zcash Ceremony, 2016. https://z.cash/blog/the-design-of-the-ceremony.html.

[2] Scott Aaronson and Yaoyun Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM*, 51(4):595–605, 2004.

[3] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, Janno Siim, and Michal Zajac. UC-Secure CRS generation for SNARKs. In *Proceedings of the 11th International Conference on Cryptology in Africa*, AFRICACRYPT '19, pages 99–117, 2019.

[4] Martin R Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Algebraic cryptanalysis of STARK-friendly designs: Application to MARVELlous and MiMC. IACR Cryptology ePrint Archive, Report 2019/419, 2019.

[5] Martin R Albrecht, Lorenzo Grassi, Léo Perrin, Sebastian Ramacher, Christian Rechberger, Dragos Rotaru, Arnab Roy, and Markus Schofnegger. Feistel structures for MPC, and more. IACR Cryptology ePrint Archive, Report 2019/397, 2019.

[6] Noga Alon, Oded Goldreich, Johan Håstad, and René Peralta. Simple construction of almost k-wise independent random variables. *Random Structures and Algorithms*, 3(3):289–304, 1992.

[7] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. IACR Cryptology ePrint Archive, Report 2019/426, 2019.

[8] Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In *Proceedings of the 39th Annual International Cryptology Conference*, CRYPTO '19, pages 269–295, 2019.

[9] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 24th ACM Conference on Computer and Communications Security*, CCS '17, pages 2087–2104, 2017.

[10] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in FOCS '92.

[11] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998. Preliminary version in FOCS '92.

[12] Tomer Ashur and Siemen Dhooghe. MARVELlous: a STARK-friendly family of cryptographic primitives. IACR Cryptology ePrint Archive, Report 2018/1098, 2018.

[13] Jean-Philippe Aumasson, Willi Meier, Raphael Phan, and Luca Henzen. *The Hash Function BLAKE*. Springer-Verlag Berlin Heidelberg, 2014.

[14] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5, 2013. `https://blake2.net/blake2.pdf`.

[15] László Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, STOC '85, pages 421–429, 1985.

[16] László Babai, Lance Fortnow, Leonid A. Levin, and Mario Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, STOC '91, pages 21–32, 1991.

[17] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991. Preliminary version appeared in FOCS '90.

[18] Barry Whitehat. Rollup, 2018. `https://github.com/barryWhiteHat/roll_up`.

[19] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear lattice-based zero-knowledge arguments for arithmetic circuits. In *Proceedings of the 38th Annual International Cryptology Conference*, CRYPTO '18, pages 669–699, 2018.

[20] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, 1993.

[21] Eli Ben-Sasson, Iddo Bentov, Alessandro Chiesa, Ariel Gabizon, Daniel Genkin, Matan Hamilis, Evgenya Pergament, Michael Riabzev, Mark Silberstein, Eran Tromer, and Madars Virza. Computational integrity with a public random string from quasi-linear pcps. In *Proceedings of the 36th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '17, pages 551–579, 2017.

[22] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed–Solomon interactive oracle proofs of proximity. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming*, ICALP '18, pages 14:1–14:17, 2018.

[23] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In *Proceedings of the 39th Annual International Cryptology Conference*, CRYPTO '19, pages 733–764, 2019.

[24] Eli Ben-Sasson, Alessandro Chiesa, Michael A. Forbes, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Zero knowledge protocols from succinct constraint detection. In *Proceedings of the 15th Theory of Cryptography Conference*, TCC '17, pages 172–206, 2017.

[25] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, Michael Riabzev, and Nicholas Spooner. Interactive oracle proofs with constant rate and query complexity. In *Proceedings of the 44th International Colloquium on Automata, Languages and Programming*, ICALP '17, pages 40:1–40:15, 2017.

[26] Eli Ben-Sasson, Alessandro Chiesa, Ariel Gabizon, and Madars Virza. Quasilinear-size zero knowledge from linear-algebraic PCPs. In *Proceedings of the 13th Theory of Cryptography Conference*, TCC '16-A, pages 33–64, 2016.

[27] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. Fast reductions from RAMs to delegatable succinct constraint satisfaction problems. In *Proceedings of the 4th Innovations in Theoretical Computer Science Conference*, ITCS '13, pages 401–414, 2013.

[28] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, and Eran Tromer. On the concrete efficiency of probabilistically-checkable proofs. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 585–594, 2013.

[29] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In *Proceedings of the 33rd Annual International Cryptology Conference*, CRYPTO '13, pages 90–108, 2013.

[30] Eli Ben-Sasson, Alessandro Chiesa, Lior Goldberg, Tom Gur, Michael Riabzev, and Nicholas Spooner. Linear-size constant-query IOPs for delegating computation. In *Proceedings of the 17th Theory of Cryptography Conference*, TCC '19, 2019.

[31] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, S&P '15, pages 287–304, 2015.

[32] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Proceedings of the 14th Theory of Cryptography Conference*, TCC '16-B, pages 31–60, 2016.

[33] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In *Proceedings of the 34th Annual International Cryptology Conference*, CRYPTO '14, pages 276–294, 2014. Extended version at `http://eprint.iacr.org/2014/595`.

[34] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von Neumann architecture. In *Proceedings of the 23rd USENIX Security Symposium*, Security '14, pages 781–796, 2014. Extended version at `http://eprint.iacr.org/2013/879`.

[35] Eli Ben-Sasson, Lior Goldberg, Swastik Kopparty, and Shubhangi Saraf. DEEP-FRI: Sampling outside the box improves soundness, 2019. ECCC TR19-044.

[36] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil Vadhan. Short PCPs verifiable in polylogarithmic time. In *Proceedings of the 20th Annual IEEE Conference on Computational Complexity*, CCC '05, pages 120–134, 2005.

[37] Eli Ben-Sasson, Oded Goldreich, Prahladh Harsha, Madhu Sudan, and Salil P. Vadhan. Robust PCPs of proximity, shorter PCPs, and applications to coding. *SIAM Journal on Computing*, 36(4):889–974, 2006.

[38] Eli Ben-Sasson, Yohay Kaplan, Swastik Kopparty, Or Meir, and Henning Stichtenoth. Constant rate PCPs for Circuit-SAT with sublinear query complexity. In *Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '13, pages 320–329, 2013.

[39] Eli Ben-Sasson, Swastik Kopparty, and Shubhangi Saraf. Worst-case to average case reductions for the distance to a code. In *Proceedings of the 33rd ACM Conference on Computer and Communications Security*, CCS '18, pages 24:1–24:23, 2018.

[40] Eli Ben-Sasson and Madhu Sudan. Robust locally testable codes and products of codes. *Random Structures and Algorithms*, 28(4):387–402, 2006.

[41] Eli Ben-Sasson and Madhu Sudan. Short PCPs with polylog query complexity. *SIAM Journal on Computing*, 38(2):551–607, 2008. Preliminary version appeared in STOC '05.

[42] Eli Ben-Sasson, Madhu Sudan, Salil Vadhan, and Avi Wigderson. Randomness-efficient low degree tests and short PCPs via epsilon-biased sets. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, STOC '03, pages 612–621, 2003.

[43] Daniel J. Bernstein and Tung Chou. Faster binary-field multiplication and faster binary-field MACs. In *Proceedings of the 21st International Conference on Selected Areas in Cryptography*, SAC '14, pages 92–111, 2014.

[44] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the sponge construction. In *Proceedings of the 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '08, pages 181–197, 2008.

[45] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 111–120, 2013.

[46] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *Proceedings of the 10th Theory of Cryptography Conference*, TCC '13, pages 315–333, 2013.

[47] Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995. Preliminary version in STOC '89.

[48] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In *Proceedings of the 17th International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '11, pages 41–69, 2011.

[49] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based SNARGs and their application to more efficient obfuscation. In *Proceedings of the 36th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT '17, pages 247–277, 2017.

[50] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Proceedings of the 35th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '16, pages 327–357, 2016.

[51] Jonathan Bootle, Andrea Cerulli, Essam Ghadafi, Jens Groth, Mohammad Hajiabadi, and Sune K. Jakobsen. Linear-time zero-knowledge proofs for arithmetic circuit satisfiability. In *Proceedings of the 23rd International Conference on the Theory and Applications of Cryptology and Information Security*, ASIACRYPT '17, pages 336–365, 2017.

[52] Sean Bowe, Ariel Gabizon, and Matthew Green. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602, 2017.

[53] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017.

[54] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019.

[55] Sean Bowe, Kobi Gurkan, Eran Tromer, Benedikt B?nz, Konstantinos Chalkias, Daniel Genkin, Jack Grigg, Daira Hopwood, Jason Law, Andrew Poelstra, abhi shelat, Muthu Venkitasubramaniam, Madars Virza, Riad S. Wahby, and Pieter Wuille. Implementation track proceeding. Technical report, ZKProof Standards, 2018. `https://zkproof.org/documents.html`.

[56] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the 39th IEEE Symposium on Security and Privacy*, S&P '18, pages 315–334, 2018.

[57] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-carrying data from accumulation schemes. Cryptology ePrint Archive, Report 2020/499, 2020.

[58] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In *Proceedings of the 39th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '20, pages 677–706, 2020.

[59] Nigel P. Byott and Robin J. Chapman. Power sums over finite subspaces of a field. *Finite Fields and Their Applications*, 5(3):254–265, July 1999.

[60] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, and Ron D. Rothblum. Fiat–Shamir from simpler assumptions. Cryptology ePrint Archive, Report 2018/1004, 2018.

[61] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM*, 51(4):557–594, 2004.

[62] David G. Cantor. On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A*, 50(2):285–300, 1989.

[63] Richard Chang, Suresh Chari, Desh Ranjan, and Pankaj Rohatgi. Relativization: a revisionistic retrospective. *Bulletin of the European Association for Theoretical Computer Science*, 47:144–153, 1992.

[64] Alessandro Chiesa, Lynn Chua, and Matthew Weidner. On cycles of pairing-friendly elliptic curves. *SIAM Journal on Applied Algebra and Geometry*, 3(2):175–192, 2019. `https://arxiv.org/abs/1803.02067`.

[65] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '20, pages 738–768, 2020.

[66] Alessandro Chiesa and Siqi Liu. On the impossibility of probabilistic proofs in relativized worlds. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference*, ITCS '20, pages 57:1–57:30, 2020.

[67] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In *Proceedings of the 17th Theory of Cryptography Conference*, TCC '19, pages 1–29, 2019. Available as Cryptology ePrint Archive, Report 2019/834.

[68] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In *Proceedings of the 1st Symposium on Innovations in Computer Science*, ICS '10, pages 310–331, 2010.

[69] Alessandro Chiesa, Eran Tromer, and Madars Virza. Cluster computing in zero knowledge. In *Proceedings of the 34th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '15, pages 371–403, 2015.

[70] Alessandro Chiesa and Zeyuan Allen Zhu. Shorter arithmetization of nondeterministic computations. *Theoretical Computer Science*, 600:107–131, 2015.

[71] Coda. The SNARK Challenge, 2019. `https://coinlist.co/build/coda`.

[72] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[73] Graham Cormode, Michael Mitzenmacher, and Justin Thaler. Practical verified computation with streaming interactive proofs. In *Proceedings of the 4th Symposium on Innovations in Theoretical Computer Science*, ITCS '12, pages 90–112, 2012.

[74] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. Geppetto: Versatile verifiable computation. In *Proceedings of the 36th IEEE Symposium on Security and Privacy*, S&P '15, pages 250–273, 2015.

[75] Ronald Cramer and Ivan Damgård. Zero-knowledge proofs for finite field arithmetic; or: Can zero-knowledge be for free? In *Proceedings of the 18th Annual International Cryptology Conference*, CRYPTO '98, pages 424–441, 1998.

[76] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. 2002.

[77] Irit Dinur. The PCP theorem by gap amplification. *Journal of the ACM*, 54(3):12, 2007.

[78] Irit Dinur and Omer Reingold. Assignment testers: Towards a combinatorial proof of the PCP theorem. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '04, pages 155–164, 2004.

[79] Sébastien Duval and Leurent Gaëtan. MDS matrices with lightweight circuits. Cryptology ePrint Archive, Report 2018/260, 2018.

[80] Electric Coin Company. Zcash Cryptocurrency, 2014. `https://z.cash/`.

[81] Funda Ergün, Ravi Kumar, and Ronitt Rubinfeld. Fast approximate probabilistically checkable proofs. *Information and Computation*, 189(2):135–159, 2004.

[82] Uriel Feige, Shafi Goldwasser, Laszlo Lovász, Shmuel Safra, and Mario Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. Preliminary version in FOCS '91.

[83] Lance Fortnow. The role of relativization in complexity theory. *Bulletin of the European Association for Theoretical Computer Science*, 52:229–244, 1994.

[84] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010.

[85] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019.

[86] Shuhong Gao and Todd Mateer. Additive fast fourier transforms over finite fields. *IEEE Transactions on Information Theory*, 56(12):6265–6272, 2010.

[87] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT '13, pages 626–645, 2013.

[88] Oded Goldreich and Madhu Sudan. Locally testable codes and PCPs of almost-linear length. *Journal of the ACM*, 53:558–655, July 2006. Preliminary version in STOC '02.

[89] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '03, pages 102–113, 2003.

[90] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *Journal of the ACM*, 62(4):27:1–27:64, 2015.

[91] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989. Preliminary version appeared in STOC '85.

[92] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge PCPs, and unconditional cryptography. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 173–190, 2010.

[93] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. IACR Cryptology ePrint Archive, Report 2019/458, 2019.

[94] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '10, pages 321–340, 2010.

[95] Jens Groth. On the size of pairing-based non-interactive arguments. In *Proceedings of the 35th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT '16, pages 305–326, 2016.

[96] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In *Proceedings of the 38th Annual International Cryptology Conference*, CRYPTO '18, pages 698–728, 2018.

[97] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In *Proceedings of the 37th Annual International Cryptology Conference*, CRYPTO '17, pages 581–612, 2017.

[98] Shay Gueron. Intel carry-less multiplication instruction and its usage for computing the GCM mode, 2011. `https://software.intel.com/en-us/articles/intel-carry-less-multiplication-instruction-and-its-usage-for-computi`

[99] Tom Gur and Ron D. Rothblum. Non-interactive proofs of proximity. In *Proceedings of the 6th Innovations in Theoretical Computer Science Conference*, ITCS '15, pages 133–142, 2015.

[100] Yuri Gurevich and Saharon Shelah. Nearly linear time. In *Logic at Botik '89, Symposium on Logical Foundations of Computer Science*, pages 108–118, 1989.

[101] Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005. Preliminary version appeared in STOC '03.

[102] Prahladh Harsha and Madhu Sudan. Small PCPs with low query complexity. *Computational Complexity*, 9(3–4):157–201, Dec 2000. Preliminary version in STACS '01.

[103] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Efficient arguments without short PCPs. In *Proceedings of the Twenty-Second Annual IEEE Conference on Computational Complexity*, CCC '07, pages 278–291, 2007.

[104] Yuval Ishai, Mohammad Mahmoody, Amit Sahai, and David Xiao. On zero-knowledge PCPs: Limitations, simplifications, and applications, 2015. Available at `http://www.cs.virginia.edu/~mohammad/files/papers/ZKPCPs-Full.pdf`.

[105] Yuval Ishai and Mor Weiss. Probabilistically checkable proofs of proximity with zero-knowledge. In *Proceedings of the 11th Theory of Cryptography Conference*, TCC '14, pages 121–145, 2014.

[106] Yael Kalai and Ran Raz. Interactive PCP. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming*, ICALP '08, pages 536–547, 2008.

[107] Joe Kilian. A note on efficient zero-knowledge proofs and arguments. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, STOC '92, pages 723–732, 1992.

[108] Rudolf Lidl and Harald Niederreiter. *Finite Fields*. Cambridge University Press, second edition edition, 1997.

[109] Sian-Jheng Lin, Tareq Y. Al-Naffouri, and Yunghsiang S. Han. FFT algorithm for binary extension finite fields and its application to Reed–Solomon codes. *IEEE Transactions on Information Theory*, 62(10):5343–5358, 2016.

[110] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han. Novel polynomial basis and its application to Reed–Solomon erasure codes. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '14, pages 316–325, 2014.

[111] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, 2003.

[112] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography*, TCC '12, pages 169–189, 2012.

[113] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In *Proceedings of the 19th International Conference on the Theory and Application of Cryptology and Information Security*, ASIACRYPT '13, pages 41–60, 2013.

[114] Richard J. Lipton. New directions in testing. In *Proceedings of a DIMACS Workshop in Distributed Computing And Cryptography*, pages 191–202, 1989.

[115] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992.

[116] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updateable structured reference strings. In *Proceedings of the 26th ACM Conference on Computer and Communications Security*, CCS '19, pages 2111–2128, 2019.

[117] Or Meir. Combinatorial PCPs with short proofs. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity*, CCC '12, 2012.

[118] Silvio Micali. Computationally sound proofs. *SIAM Journal on Computing*, 30(4):1253–1298, 2000. Preliminary version appeared in FOCS '94.

[119] Thilo Mie. Short PCPPs verifiable in polylogarithmic time with o(1) queries. *Annals of Mathematics and Artificial Intelligence*, 56:313–338, 2009.

[120] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[121] O(1) Labs. Coda Cryptocurrency, 2017. `https://codaprotocol.com/`.

[122] eBACS: ECRYPT Benchmarking of Cryptographic Systems. Measurements of hash functions, indexed by machine, 2017.

[123] Christos H. Papadimitriou and Mihalis Yannakakis. A note on succinct representations of graphs. *Information and Control*, 71(3):181–185, 1986.

[124] Brian Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, Oakland '13, pages 238–252, 2013.

[125] Rafael Pass. On deniability in the common reference string and random oracle model. In *Proceedings of the 23rd Annual International Cryptology Conference*, CRYPTO '03, pages 316–337, 2003.

[126] Alexander Polishchuk and Daniel A. Spielman. Nearly-linear size holographic proofs. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, STOC '94, pages 194–203, 1994.

[127] Omer Reingold, Ron Rothblum, and Guy Rothblum. Constant-round interactive proofs for delegating computation. In *Proceedings of the 48th ACM Symposium on the Theory of Computing*, STOC '16, pages 49–62, 2016.

[128] J. M. Robson. An O(T log T) reduction from RAM computations to satisfiability. *Theoretical Computer Science*, 82(1):141–149, May 1991.

[129] Noga Ron-Zewi and Ron D. Rothblum. Local proofs approaching the witness length, 2019.

[130] Guy N. Rothblum, Salil P. Vadhan, and Avi Wigderson. Interactive proofs of proximity: delegating computation in sublinear time. In *Proceedings of the 45th ACM Symposium on the Theory of Computing*, STOC '13, pages 793–802, 2013.

[131] SCIPR Lab. libsnark: a C++ library for zkSNARK proofs, 2014.

[132] SCIPR Lab. libiop: C++ library for IOP-based zkSNARKs, 2019.

[133] SCIPR Lab. A Rust library for decentralized private computation, 2019.

[134] SCIPR Lab. A rust library for the marlin preprocessing zksnark, 2019.

[135] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. Cryptology ePrint Archive, Report 2019/550, 2019.

[136] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J. Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th EuoroSys Conference*, EuroSys '13, pages 71–84, 2013.

[137] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, 1992.

[138] Amir Shpilka and Amir Yehudayoff. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 2010.

[139] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996. Preliminary version appeared in STOC '95.

[140] StarkWare & 0x. StarkDEX, 2019. `https://www.starkdex.io/`.

[141] Justin Thaler. Time-optimal interactive proofs for circuit evaluation. In *Proceedings of the 33rd Annual International Cryptology Conference*, CRYPTO '13, pages 71–89, 2013.

[142] Justin Thaler. A note on the GKR protocol. `http://people.cs.georgetown.edu/jthaler/GKRNote.pdf`, 2015.

[143] Justin Thaler, Mike Roberts, Michael Mitzenmacher, and Hanspeter Pfister. Verifiable computation with massively parallel interactive proofs. *CoRR*, abs/1202.1350, 2012.

[144] Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In *Proceedings of the 5th Theory of Cryptography Conference*, TCC '08, pages 1–18, 2008.

[145] Riad S. Wahby, Max Howald, Siddharth J. Garg, Abhi Shelat, and Michael Walfish. Verifiable ASICs. In *Proceedings of the 37th IEEE Symposium on Security and Privacy*, S&P '16, pages 759–778, 2016.

[146] Riad S. Wahby, Ye Ji, Andrew J. Blumberg, Abhi Shelat, Justin Thaler, Michael Walfish, and Thomas Wies. Full accounting for verifiable outsourcing. In *Proceedings of the 24th ACM Conference on Computer and Communications Security*, CCS '17, pages 2071–2086, 2017.

[147] Riad S. Wahby, Srinath Setty, Zuocheng Ren, Andrew J. Blumberg, and Michael Walfish. Efficient RAM and control flow in verifiable outsourced computation. In *Proceedings of the 22nd Annual Network and Distributed System Security Symposium*, NDSS '15, 2015.

[148] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *Proceedings of the 39th IEEE Symposium on Security and Privacy*, S&P '18, pages 926–943, 2018.

[149] Michael Walfish and Andrew J. Blumberg. Verifying computations without reexecuting them. *Communications of the ACM*, 58(2):74–84, January 2015.

[150] Hoeteck Wee. On round-efficient argument systems. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, ICALP '05, pages 140–152, 2005.

[151] Tiancheng Xie, Jiaheng Zhang, Yupeng Zhang, Charalampos Papamanthou, and Dawn Song. Libra: Succinct zero-knowledge proofs with optimal prover computation. In *Proceedings of the 39th Annual International Cryptology Conference*, CRYPTO '19, pages 733–764, 2019.

[152] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In *Proceedings of the 39th Annual International Cryptology Conference*, CRYPTO '19, pages 239–268, 2019.

[153] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases. In *Proceedings of the 38th IEEE Symposium on Security and Privacy*, S&P '17, pages 863–880, 2017.

[154] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. A zero-knowledge version of vsql. Cryptology ePrint Archive, Report 2017/1146, 2017.

[155] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vRAM: Faster verifiable RAM with program-independent preprocessing. In *Proceedings of the 39th IEEE Symposium on Security and Privacy*, S&P '18, pages 908–925, 2018.

[156] ZKP Standards. Zero knowledge proof standardization, 2017. `https://zkproof.org/`.

Appendix A

Appendix

A.1 Proof of Lemma 3.4.4

Definition A.1.1. *For a field $\mathbb{F}$ of characteristic p, the generalized derivative of a function f in a direction $a \in \mathbb{F}$ is $\Delta_a(f) := \sum_{b\in\mathbb{F}_p} f(X + ba)$. For $a_1, \ldots, a_k \in \mathbb{F}$, we inductively define $\Delta_{a_1,\ldots,a_k}(f) := \Delta_{a_1}(\Delta_{a_2,\ldots,a_k}(f))$.*

Note that if $\mathbb{F}$ has characteristic 2 then this coincides with the directional derivative. If H is a subspace of $\mathbb{F}$ with basis $a_1, \ldots, a_n$ then for any $a_0 \in \mathbb{F}$,

$$\Delta_{a_1,\ldots,a_k}(f)(a_0) = \sum_{a\in H_0} f(a_0 + a) \ . \tag{A.1}$$

For a natural number $c = \sum_{i=0}^{k} c_i p^i$, $0 \le c_i < p$, let $\mathsf{wt}(c) = \sum_{i=0}^{k} c_i$.[1] For a polynomial $P(X) = \sum_{j\ge 0} \alpha_j X^j$ define $\mathsf{wt}(P) := \max\{\mathsf{wt}(j) : \alpha_j \neq 0\}$.

Claim A.1.2. *For any polynomial $P \in \mathbb{F}[X]$ and any $a \in \mathbb{F}$,*

$$\mathsf{wt}(\Delta_a(P)) \le \max(\mathsf{wt}(P) - (p-1), 0) \ .$$

Moreover, if $\mathsf{wt}(P) < p - 1$, then $\Delta_a(P)$ is identically zero.

Proof. By linearity of Δ_a, it suffices to prove the claim for a single monomial; that is, $P(X) = X^c$ for some integer $c \ge 0$. Let $c = \sum_{i=0}^{k} c_i p^i$ be the p-ary expansion of c for some integer k. For a natural number $d = \sum_{i=0}^{k} d_i p^i$ we write $d \le_p c$ if $d_i \le c_i$ for all i.

$$\begin{aligned}
\Delta_a(X^c) &= \sum_{b\in\mathbb{F}_p} (X + ba)^c = \sum_{b\in\mathbb{F}_p} \sum_{d=0}^{c} \binom{c}{d} X^d b^{c-d} a^{c-d} \\
&= \sum_{d=0}^{c} \binom{c}{d} X^d a^{c-d} \Big(\sum_{b\in\mathbb{F}_p} b^{c-d} \Big) = \sum_{d=0}^{c} \binom{c}{d} X^d a^{c-d} \Big(\sum_{b\in\mathbb{F}_p} b^{\sum_{i=0}^{k}(c_i - d_i)p^i} \Big) \\
&= \sum_{d=0}^{c} \binom{c}{d} X^d a^{c-d} \Big(\sum_{b\in\mathbb{F}_p} b^{\sum_{i=0}^{k}(c_i - d_i)} \Big) = \sum_{d\le_p c} \binom{c}{d} X^d a^{c-d} \cdot \Big(\sum_{b\in\mathbb{F}_p} b^{\mathsf{wt}(c) - \mathsf{wt}(d)} \Big) \ ,
\end{aligned}$$

[1] This quantity is called the 'q-ary digit sum' in [59].

where in the penultimate equality we used that $b^{p^i} = b$ for $b \in \mathbb{F}_p$, and in the last equality we used that $\binom{c}{d} \equiv 0 \pmod{p}$ unless $d \leq_p c$. Recall that for $0 \leq m < p-1$, $\sum_{b \in \mathbb{F}_p} b^m = 0$. Hence the terms in the above sum where $\mathtt{wt}(c) - \mathtt{wt}(d) < p-1$ all vanish. Any remaining terms thus have weight at most $\mathtt{wt}(c) - (p-1)$. In particular, if $\mathtt{wt}(c) < p-1$ then all terms vanish. □

Lemma A.1.3. *Let $\mathbb{F}$ be a field of characteristic p, and let $P \in \mathbb{F}[X]$ have degree less than $p^k - 1$. Then for any $a_1, \ldots, a_k \in \mathbb{F}$, $\Delta_{a_1,\ldots,a_k}(P)$ is identically zero.*

Proof. We have $\mathtt{wt}(P) < (p-1)k$. By Claim A.1.2, $\mathtt{wt}(\Delta_{a_2,\ldots,a_k}(P)) < p-1$, and so $\Delta_{a_1,\ldots,a_k}(P)$ is identically zero. □

Proof of Lemma 3.4.4. For some $a_0, a_1, \ldots, a_k \in \mathbb{F}$, $H = a_0 + H_0$ where H_0 is the linear subspace with basis $a_1, \ldots, a_k$. By Eq. (A.1) and Lemma A.1.3 we conclude $\sum_{a \in H} g(a) = \Delta_{a_1,\ldots,a_k}(g)(a_0) = 0$. □

A.2 Proof of Lemma 3.4.5

This proof is due to Craig Gentry; I merely record it below. For convenience, we restate the lemma.

Lemma A.2.1. *If H is an affine subspace of $\mathbb{F}$, then $\sum_{a \in H} a^{|H|-1}$ equals the linear term of $\mathbb{Z}_H$.*

Proof. Let $e_k(x_1, \ldots, x_n)$ be the k-th elementary symmetric polynomial in n variables. Note that since e_k is symmetric it is well-defined to write $e_k(S)$ for a set S.

Recall that $\mathbb{Z}_H(X) = \prod_{a \in H}(X - a)$. Since $0 \in H$, the linear term of $\mathbb{Z}_H$ is equal to $\prod_{a \in H \setminus \{0\}} a = e_{|H|-1}(H \setminus \{0\})$. By Newton's identities, it holds that

$$(|H|-1)e_{|H|-1}(H \setminus \{0\}) = \sum_{i=1}^{|H|-1} (-1)^{i-1} e_{|H|-i-1}(H \setminus \{0\}) \cdot \sum_{a \in H} a^i \ .$$

By Lemma 3.4.4, it holds that for all $i < |H| - 1$, $\sum_{a \in H} a^i = 0$. Hence

$$e_{|H|-1}(H \setminus \{0\}) = (-1)^{|H|-1} \sum_{a \in H} a^{|H|-1} \ ,$$

where we used that $|H| - 1 \equiv -1 \pmod{\mathrm{char}(\mathbb{F})}$. The lemma follows by noting that in all fields $\mathbb{F}$ of positive characteristic, $(-1)^{t-1} = 1$ when $\mathrm{char}(\mathbb{F})$ divides t. □

A.3 Additional comparisons

We provide additional comparisons across Ligero, Stark, and Aurora: in Appendix A.3.1 we compare the low-degree tests that they rely on, and in Appendix A.3.2 we compare their underlying IOP protocols.

A.3.1 Comparison of the LDTs in Ligero, Stark, and Aurora

A key ingredient in Ligero [9], Stark [23], and Aurora (this work) are low-degree tests (LDTs). Formally, each of these systems relies on an IOPP for the Reed–Solomon relation (see Section 2.3.1.1). The LDT is then generically "lifted" to an LDT for the interleaved Reed–Solomon code (see Section 2.1.1), by taking a random linear combination as in Section 3.7. Below (and in Fig. A.1) we discuss aspects of the LDTs underlying these systems that are important in the comparison in Appendix A.3.2.

Direct LDT. Ligero uses a *direct* LDT: the verifier is given oracle access to a function $f\colon L \to \mathbb{F}$, receives from the prover $a_0, \ldots, a_{\rho|L|-1} \in \mathbb{F}$ (allegedly, coefficients of the polynomial $\hat{f}$ obtained by interpolating f), and checks that f and $\sum_{i=0}^{\rho|L|-1} a_i X^i$ agree at a random point of $|L|$. If f is δ-far from $\mathrm{RS}\,[L,\rho]$ then the verifier accepts with probability at most $1-\delta$. This probability can be reduced to $(1-\delta)^t$ via t independent checks. Overall, the verifier queries f at t points, and reads $\rho|L|$ field elements sent by the prover. One should think of t as much less than $\rho|L|$, which facilitates lifting to an LDT for the interleaved Reed–Solomon code.

FRI LDT. Stark and Aurora use FRI [22], a LDT in which the verifier is given oracle access to a function $f\colon L \to \mathbb{F}$ and, in each of a sequence of rounds, sends a random field element to the prover, who replies with an oracle; at the end of the interaction, the verifier makes a certain number of queries to f and the oracles, and then either accepts or rejects. (The domain L here is an additive or multiplicative coset in $\mathbb{F}$ whose order is a power of 2.) In more detail, given a *localization parameter* $\eta \in \mathbb{N}$, the number of rounds is $\frac{\log \rho|L|}{\eta}$, and in the i-th round the prover sends an oracle over a domain of size $|L|/2^{i\eta}$; thus, the total number of elements sent across all oracles is less than $\sum_{i=1}^{\infty} |L|/2^{i\eta} = |L|/(2^\eta+1)$. After the interaction, the verifier queries f at a point, and every other oracle at $2^\eta - 1$ points; given the corresponding answers, the verifier performs $O(2^\eta \log \rho|L|)$ arithmetic operations, and then accepts or rejects. If f is δ-far from $\mathrm{RS}\,[L,\rho]$ then the verifier, with probability at most $\varepsilon_{\mathsf{i}}^{\mathrm{FRI}}(\mathbb{F},L)$ over its random messages to the prover, will accept with probability greater than $\varepsilon_{\mathsf{q}}^{\mathrm{FRI}}(L,\rho,\delta)$. In [22] it is proved that $\varepsilon_{\mathsf{i}}^{\mathrm{FRI}}(\mathbb{F},L) = 3|L|/|\mathbb{F}|$ and $\varepsilon_{\mathsf{q}}^{\mathrm{FRI}}(L,\rho,\delta) = 1-\min\{\delta, (1-3\rho-2^\eta|L|^{-1/2})/4\}$. In [39] this was improved to $\varepsilon_{\mathsf{i}}^{\mathrm{FRI}}(\mathbb{F},L) = 2\log|L|/\epsilon^3|\mathbb{F}|$ and $\varepsilon_{\mathsf{q}}^{\mathrm{FRI}}(L,\rho,\delta) = 1-\min\{\delta, J_\epsilon(J_\epsilon(1-\rho))\}+\epsilon\log|L|$ for any $\epsilon>0$, where $J_\epsilon(x) := 1-\sqrt{1-x(1-\epsilon)}$. In [22] it is conjectured that the best possible values are $\varepsilon_{\mathsf{i}}^{\mathrm{FRI}}(\mathbb{F},L) = 2^\eta \log^2(|L|)/\epsilon\eta^2|\mathbb{F}|$ and $\varepsilon_{\mathsf{q}}^{\mathrm{FRI}}(L,\rho,\delta) = 1-\delta(1-\epsilon)$ for any $\epsilon>0$.

A.3.2 Comparison of the IOPs in Ligero, Stark, and Aurora

Each of Ligero [9], Stark [23], and Aurora (this work) is an IOP (satisfying zero knowledge and proof of knowledge) that is compiled into a zkSNARK via a transformation of Ben-Sasson et al. [32]. Comparing these zkSNARKs (essentially) reduces to comparing the underlying IOPs, which we do below.

Construction blueprint. The IOPs in the aforementioned systems can *all* be viewed as combining an encoded IOP (as defined in Section 2.3.3) and a low-degree test (as defined in Section 2.3.1.1), via the transformation described in Section 3.7. Informally, this transformation invokes the low-degree test on a suitable random linear combination of the oracles sent by the encoded IOP prover (more generally, of "virtual" oracles implied by these), thereby ensuring

LDT	number of queries to f	number of queries to aux oracles	soundness error
direct	t	$\rho\|L\|$	$(1-\delta)^t$
FRI	t	$t \cdot (2^\eta - 1) \cdot \frac{\log \rho\|L\|}{\eta}$	$\varepsilon_{\mathsf{i}}^{\mathrm{FRI}}(\mathbb{F}, L) + \varepsilon_{\mathsf{q}}^{\mathrm{FRI}}(L, \rho, \delta)^t$

Figure A.1: Parameters of the direct low-degree test and FRI low-degree test when invoked on a function $f\colon L \to \mathbb{F}$ that is δ-far from $\mathrm{RS}[L, \rho] \subseteq \mathbb{F}^L$. Note that δ always lies in $[0, 1-\rho]$.

IOP	relation	number of queries
Stark	APR	$\mathsf{q}_{\mathrm{FRI}}\big(\|\mathcal{R}\|+1,\, 4(\|H\|+\mathsf{b}),\, \rho\big)$ $+\,\mathsf{q}_{\mathrm{FRI}}\big(\|\mathcal{N}\|+1,\, D(\|H\|+\mathsf{b}),\, \rho\big)$
Ligero	R1CS	$\mathsf{q}_{\mathrm{DIR}}\big(4(h+1),\, 2m/h,\, \rho\big)$
Aurora	R1CS	$\mathsf{q}_{\mathrm{FRI}}\big(5,\, 2m+2\mathsf{b},\, \rho\big)$

Figure A.2: Aspects of the IOPs underlying Stark, Ligero, and Aurora.

that the codeword obtained by stacking these oracles is close to the interleaved Reed–Solomon code (more generally, a codeword obtained by applying a transformation to these oracles is close to the interleaved Reed–Solomon code); one can then reduce to soundness of the encoded IOP.

For a given soundness error, the query complexity of an IOP constructed via the blueprint above is determined by the query complexity of the underlying low-degree test, while (typically) the prover and verifier complexities are dominated by the encoded IOP's prover and verifier complexities.

The three IOPs. In light of the foregoing blueprint, we describe the differences across the three IOPs by discussing the differences across the respective encoded IOPs and low-degree tests (see Fig. A.2). Recall that b denotes the query bound for zero knowledge (as defined in Section 2.3.2); the bound is later set to equal the number of queries of the honest verifier. Moreover, for notational simplicity, below we use $\mathsf{q}\big(k,\, d,\, \rho\big)$ to denote the query complexity of a low-degree test invoked on a function $f^*\colon L \to \mathbb{F}$ derived entry-wise from k oracles $f_i\colon L \to \mathbb{F}$ sent by the encoded IOP prover, with each oracle (allegedly) having degree less than $d = \rho|L|$; using a low-degree test in this way follows the general paradigm described in Section 2.1.1.

- *The IOP in Aurora.* The IOP in Aurora is obtained by combining an encoded IOP for R1CS (described in Section 3.6) and the FRI low-degree test. Given an R1CS instance with m constraints, the IOP invokes the low-degree test on 5 oracles having maximal degree $2m + 2\mathsf{b}$, resulting in $\mathsf{q}_{\mathrm{FRI}}\big(5,\, 2m+2\mathsf{b},\, \rho\big)$ queries.

- *The IOP in Ligero.* The IOP in Ligero (adapted for R1CS) is obtained by combining an encoded IOP for R1CS and a direct low-degree test (see Appendix A.3.1). Given an R1CS instance with m constraints and for a parameter $h \approx \sqrt{m}$, the IOP invokes the low-degree test on $4(h+1)$ oracles of maximal degree $2m/h$, resulting in $\mathsf{q}_{\mathrm{DIR}}\big(4(h+1),\, 2m/h,\, \rho\big)$ queries.

- *The IOP in Stark.* The IOP in Stark natively supports *Algebraic Placement and Routing* (APR), which is the following problem: given a finite field $\mathbb{F}$, subset $H \subseteq \mathbb{F}$, algebraic

registers $\mathcal{R}$, neighbors $\mathcal{N}$, and set of polynomial constraints $\mathcal{C}$, are there functions $w = (w_i \colon H \to \mathbb{F})_{i \in \mathcal{R}}$ such that for every element $\alpha \in H$ and every constraint $c \in \mathcal{C}$ it holds that $c\big(\alpha, (w_i(f(\alpha)))_{(i,f) \in \mathcal{N}}\big) = 0$? (See [23] for details.)

The IOP in Stark is obtained by combining an encoded IOP for APR and the FRI low-degree test. The latter is used twice: once on $|\mathcal{R}| + 1$ oracles of maximal degree $4(|H| + \mathsf{b})$; once on $|\mathcal{N}| + 1$ oracles of maximal degree $D(|H| + \mathsf{b})$. This results in $\mathsf{q}_{\text{FRI}}\big(|\mathcal{R}| + 1,\ 4(|H| + \mathsf{b}),\ \rho\big) + \mathsf{q}_{\text{FRI}}\big(|\mathcal{N}| + 1,\ D(|H| + \mathsf{b}),\ \rho\big)$ queries.